[redacted]

51

Math

8

Report No. UIUCDCS-R-77-851

UILU-ENG 77 1705

TOPICS IN MOSFET NETWORK DESIGN

by

Jay Niel Culliney

February 1977

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. UIUCDCS-R-77-851

TOPICS IN MOSFET NETWORK DESIGN

by

Jay Niel Culliney

February 1977

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

his work was supported in part by the National Science Foundation under Grant o. DCR73-03421 and was submitted in partial fulfillment of the requirements or the degree of Doctor of Philosphy in Computer Science, January 1977.

This thesis is
dedicated to my wife Yuko
without whose help and great patience this work
could not have been completed.

ACKNOWLEDGMENT

The author wishes to thank his advisor, Professor S. Muroga, for his invaluable guidance during the preparation of this thesis and during the several preceding years of research, and also for his careful reading and constructive criticism of the original manuscript. The author also wishes to thank Dr. H.C. Lai and Dr. Y. Kambayashi for their helpful discussions and constructive comments. The author is grateful to Zigrida Arbatsky for her typing of the manuscript. The financial support of the Department of Computer Science and the National Science Foundation under Grant No. DCR73-03421 is also acknowledged.

TABLE OF CONTENTS

SECTION Page

SECTION Page

1. INTRODUCTION

In recent years, MOS technology has emerged as one of the most important technologies for large-scale integration (LSI). In addition to the practical interest in MOS, it has been found that MOS cells lend themselves to theoretical treatment for certain common network synthesis problems.

Theoretically, an MOS cell can be constructed to realize any negative function. This fact leads to the convenient representation of a general MOS cell by a negative gate.

The problem of synthesizing a 2-level network with the minimum number of negative gates was first solved by T. Ibaraki and S. Muroga.[IM 71] This work was then extended by T. Ibaraki[Iba 71] to minimize either or both of the numbers of gates and connections in a 2-level network of negative gates, although the method involves the solution of covering problems.

Next, results were obtained concerning the problem of synthesizing multiple-level networks with minimum numbers of negative gates. Nakamura, Tokura, and Kasami,[NTK 72] pointed out that the older problem of minimizing the number of inverters (NOT gates) in a network, which was solved by both Markov [Mar 58] and Muller,[Mul 58] is very closely related to the optimal multiple-level negative gate network synthesis problem, and they presented algorithms for the latter problem. Independently, T. K. Liu[Liu 72] solved the same problem and also gave another approach to the optimal 2-level network synthesis problem.

Later, H. C. Lai[Lai 76] extended these results and presented algorithms which synthesize multiple-level MOS cell networks with minimum numbers of cells which are, in addition, 'irredundant' in the sense that no FET's can be removed from the MOS cells of the network without altering the desired outputs of the network. Networks produced by former methods were often not irredundant.

T. Shinozaki[Shi 72] and K. Yamamoto[Yam 76] produced and documented computer programs implementing MOS network synthesis algorithms of T. K. Liu and H. C. Lai, respectively.

This thesis will further explore topics in MOS network synthesis. Section 2 will provide the background for later discussion by presenting basic definitions, theorems, and notation. The operation of an MOS cell will also be explained here. Section 3 will present properties of optimal networks of MOS cells or negative gates and some properties of interest to logic designers which apply to non-optimal networks as well. These results are useful in assisting (optimal and/or non-optimal) network synthesis efforts and in facilitating the recognition of non-optimal networks.

After briefly reviewing several existing optimal network synthesis methods in Section 4, new methods based on a redefined concept of 'clusters' are proposed to obtain improved synthesis results. Section 5 discusses transformation of MOS networks which are based only upon the configurations of the network and the individual MOS cells. These are suitable for use without the aid of a computer. An important application of these transformations would be in the transformation of networks of overly complex MOS cells (as often result from optimal MOS network synthesis algorithms) into networks of simpler MOS cells, practical for actual implementation.

A new approach to negative gate network synthesis is presented in Section 6. This approach basically consists of procedures for the transformation and reduction (referred to as 'transduction procedures') of existing networks in order to obtain simplified networks. This differs from the transformations in Section 5 in that these transduction procedures consider the specific functions of the negative gates in a network, as well as the configuration of the network itself. Thus, these procedures are both more powerful and more complex than the transformations of Section 5. Due to this complexity, implementations of the transduction procedures as computer programs are required for the solution of all but the simplest problems. Section 6 also discusses such implementations and gives examples of synthesis results obtained.

Section 7 presents a method for the generation of reduced test sets for the diagnosis of irredundant MOS networks synthesized by Lai's algorithm. The generation is most conveniently accomplished during the process of synthesizing the irredundant networks.

2. BASIC DEFINITIONS

To facilitate later discussion, some of the basic definitions, symbols, and concepts common to topics in succeeding sections are given at this point. A few closely related basic theorems are also included which are discussed, or related to those discussed, in existing literature, mainly in [Gil 54], [IM 71], [NTK 72], and [Lai 76]. Further definitions and basic theorems more pertinent to a topic discussed in a particular section are given in the respective section.

A typical MOS (Metal Oxide Semiconductor) cell is shown in Fig. 2.1. It consists of several interconnected field-effect transistors (FET's) of which one is called the load and the rest, often connected in a series and parallel fashion, constitute what is called the driver. The load FET is always conducting and provides a permanent connection between the output terminal and the voltage supply through the load resistance which it represents. The FET network constituting the driver section operates exactly like a relay-contact network where each FET corresponds to a 'normally open' relay contact: it is conductive when its input is a logical '1' (represented by a voltage near the level of the supply voltage), and it is non-conductive when its input is a logical '0' (a voltage near the level of ground). A proper combination of input signals to the driver (e.g., A,D = '1' in Fig. 2.1) creates a conducting path between the output terminal and ground. Because of the relatively high resistance represented by the load and the relatively low resistance represented by the driver, this results in a logical '0' at the output terminal (i.e., a voltage close to ground). When a particular combination of input signals results in no conducting path from output terminal to ground (e.g., D,E = '0'), the

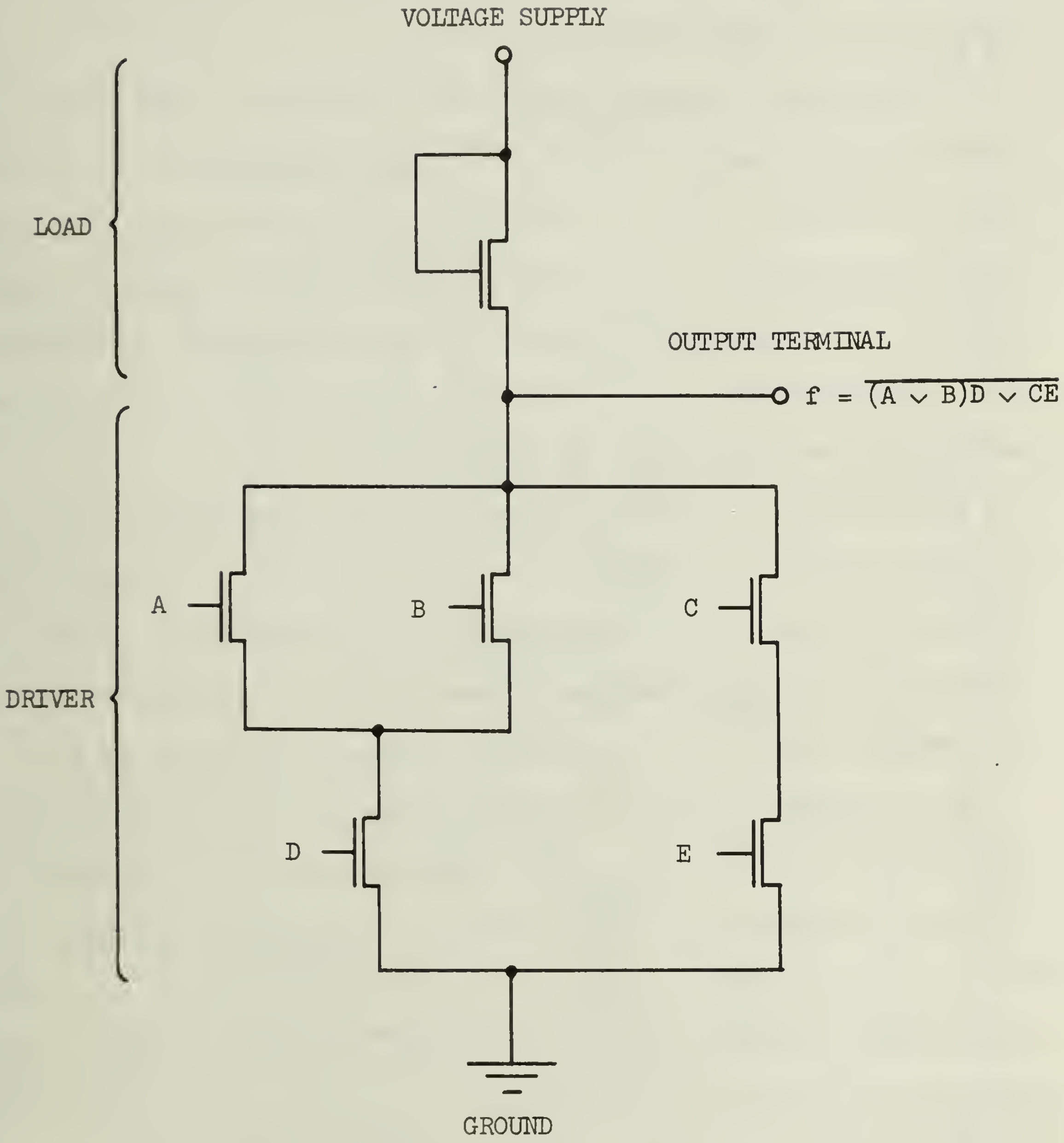


Fig. 2.1 A typical MOS cell.

output is a logical '1' since it is at a voltage level almost identical to that of the voltage supply. Thus the MOS cell of Fig. 2.1 can be seen to realize the function $\overline{(A \vee B)D \vee CE}$.

A <u>two-terminal switching network</u> is a two-terminal graph defined in graph theory (see, for example, [Har 65]) with edges consisting of branch-type elements (in this case, FET's). Discussion in this work is restricted, except where otherwise stated, to two-terminal series-parallel networks to simplify analyses. The <u>transmission function</u>, f_N, of a two-terminal network N is a Boolean function which is '1' if and only if there is a closed (conductive) path between the terminals of the network.

The driver of an MOS cell together with its corresponding output terminal and ground connection constitute a two-terminal network. By the preceding discussion of the operation of an MOS cell, if the transmission function of an MOS cell's driver is f_N, the function represented by the voltage level of the output terminal, referred to as the <u>output function of the cell</u>, is $\overline{f}_N$.

Fig. 2.2 shows a typical MOS network (the network shown happens to be a 1-bit adder). In order to facilitate later discussion, three types of connections are distinguished in an MOS network. An <u>intercell connection</u> is a connection from an external variable, x_i, or the output terminal of a cell, g_i, to the driver of another cell, g_j, considered as a unit. In other words, regardless of the number of individual FET's g_i (or x_i) may be connected to in the driver of g_j, there is considered to be only one intercell connection from g_i (or x_i) to g_j. There are 10 intercell connections in the network of Fig. 2.2, three to each of cells g_1 and g_2, and four to cell g_3. A second type of connection is the <u>intracell connection</u>. This refers to connections among the

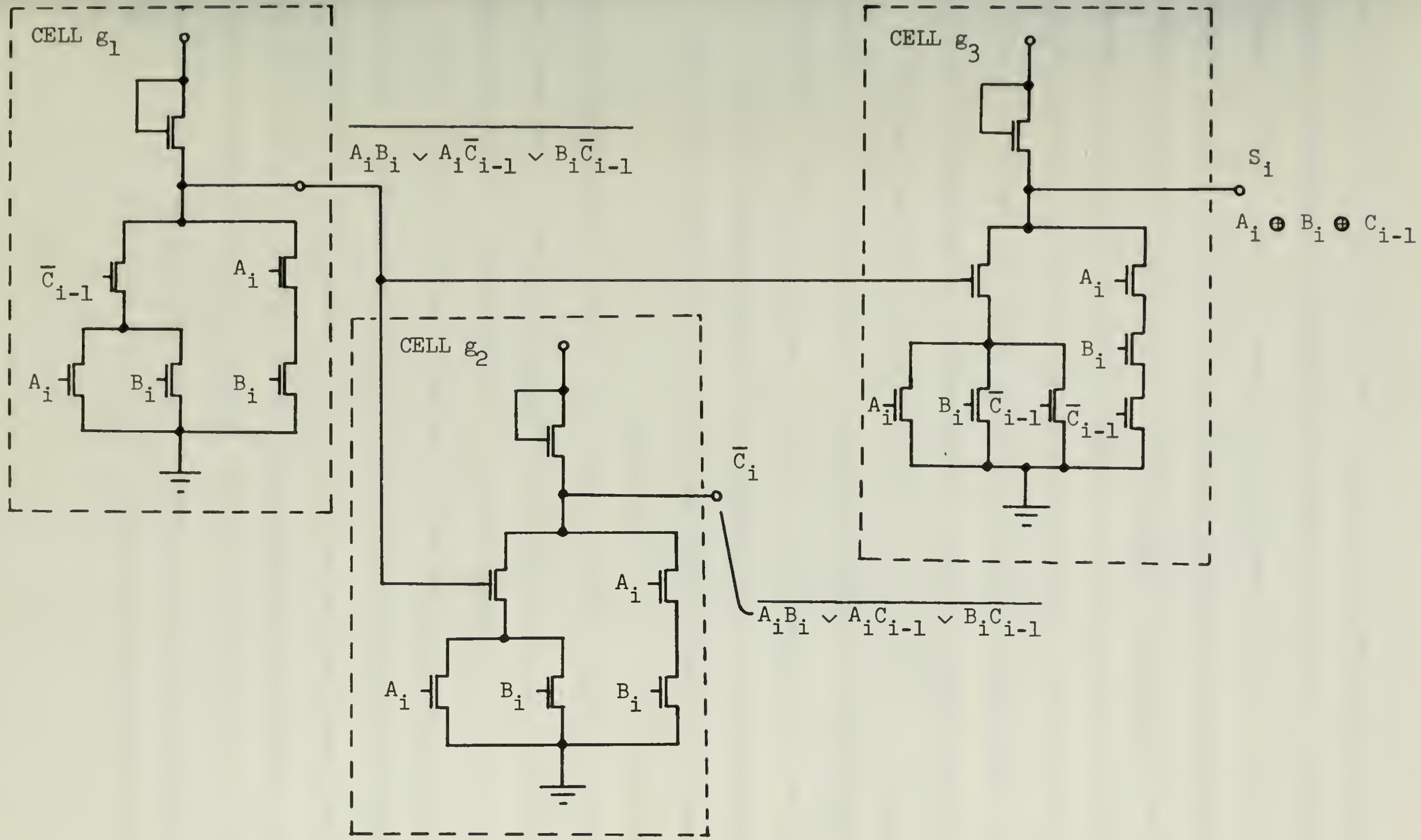


Fig. 2.2 Example of MOS cell network employing three cells. (Network is actually a 1-bit adder in which A_i and B_i are the bits to be added and C_i is the carry from the adder for the i-th bit position.)

output terminal, driver FET's, and ground of an individual cell. For the purposes of this work, the number of such intraconnections will not be as important as the number of FET's connected or the manner in which they are connected (i.e., in series, in parallel). Connections of the third type are simply referred to as connections. Each connection from a cell or external variable to an individual FET is counted as a separate connection. Hence the number of connections in an MOS cell network is equal to the number of driver FET's in the network. The number of connections in the network of Fig. 2.2 is 17.

In a similar sense, two types of networks can be distinguished as well as two types of paths through the networks. 'Network' can refer to either a network of MOS cells or a network of FET's. These have not been given special names since the intended meaning should always be clear from the context in which the word appears. A path through a two-terminal network of FET's is just a sequence of connected FET's leading from one terminal to the other. Other types of paths will be defined shortly.

An MOS cell g_i (or external variable x_i) is said to be an immediate predecessor of a cell g_j if there is an intercell connection from the output terminal of g_i (from x_i) to the driver of g_j. In this case, cell g_j is also said to be an immediate successor of cell g_i (external variable x_i).

If there exists a sequence of cells $g_{i_1}, g_{i_2}, \ldots, g_{i_s}$, $s \geq 2$, such that each g_{i_j} is an immediate predecessor of $g_{i_{j+1}}$, then: g_{i_s} is called a successor of g_{i_1}, g_{i_1} is called a predecessor of g_{i_s}, and a path is said to exist from cell g_{i_1} to g_{i_s}. These definitions are extended to allow external variables as possible predecessors (i.e., substitute external variable x_i for cell g_{i_1} in the preceding group of definitions).

Let $IP(g_i)$, $IS(g_i)$, $P(g_i)$, and $S(g_i)$ denote, respectively, the sets of immediate predecessors, immediate successors, predecessors, and successors of cell g_i.

A network of MOS cells in which no cell is a successor (or predecessor) of itself is called a loop-free or feed-forward network. In this work, only loop-free networks will be considered.

Next, notation, definitions, and basic theorems pertaining to 'negative functions' are developed. The important relation between these 'negative functions' and the MOS cell networks just described will be discussed later.

Switching functions of n variables, $x_1,\ldots,x_n$, will be considered.

Let V_n be the set of all n-dimensional binary (0 or 1) vectors. Given two vectors $A = (a_1,\ldots,a_n)$ and $B = (b_1,\ldots,b_n)$ in V_n:

$A = B$ denotes the fact that $a_i = b_i$ for every $i = 1,\ldots,n$;

$A < B$ denotes the fact that $a_i \leq b_i$ for every $i = 1,\ldots,n$, and $a_i < b_i$ for at least one i.

If none of the relations $A > B$, $A = B$, or $A < B$ holds between A and B, vectors A and B are said to be incomparable. For convenience, let $\vec{0}$ and $\vec{1}$ denote the special vectors $(0,\ldots,0)$ and $(1,\ldots,1)$, respectively.

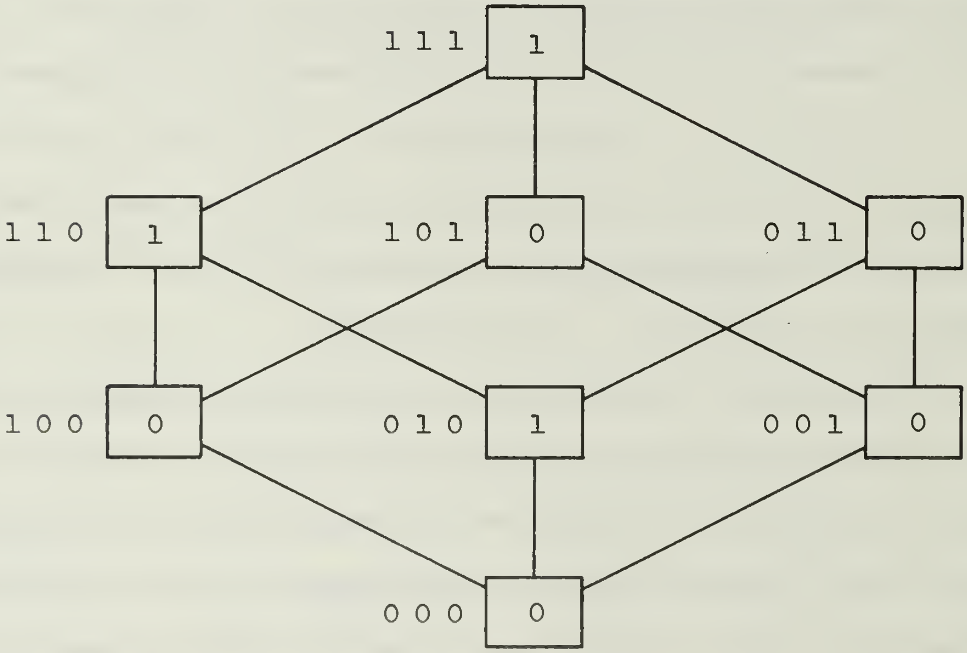
There are many possible ways in which to represent an n-variable switching function. Different representations often have respective unique advantages for various purposes. In addition to the familiar Boolean representation (see Fig. 2.3(a)), the well-known truth table representation (see Fig. 2.3(b)) is also often useful. In the theorems and algorithms of later sections, the truth table representation and what can be considered to be a variation of it, the labelled n-dimensional cube, are invaluable.

$$f = x_1x_2 \vee x_2\bar{x}_3$$

(a) Function f expressed in Boolean representation.

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

(b) Function f expressed in truth table representation.



(c) Function f expressed in labelled n-cube representation. (Labels are the numbers inside boxes.)

Fig. 2.3 Different representations of a function f.

An <u>n-dimensional cube</u> (abbreviated to <u>n-cube</u>), denoted C_n, is defined as follows:

(1) Each vector in V_n is represented as a distinct vertex in C_n. Henceforth, a vertex in C_n may be referred to by the vector it represents (e.g., 'vertex (101)').

(2) There is an edge between two vertices A and B if and only if A differs from B in exactly one component (i.e., A and B are of Hamming distance 1).

(3) An edge between vertices A and B is directed from A to B if and only if A > B and is denoted by $\overrightarrow{AB}$. A 3-cube, C_3, is shown in Fig. 2.4.

The <u>weight</u> of a vertex A, denoted w(A), is defined as the number of '1' components in vector A.

A vector A in V_n (corresponding to a vertex in C_n) can be considered to be a combination of values of the n variables $x_1,\ldots,x_n$ where the i-th component of A represents the value of variable x_i. Such a vector A is called an <u>input vector</u> for a function f of $x_1,\ldots,x_n$.

A <u>completely specified function</u> of n variables, $x_1,\ldots,x_n$, is one for which a value is specified for every input vector $A \in V_n$. An <u>incompletely specified function</u> of n variables, $x_1,\ldots,x_n$, is one for which no value is specified for at least one input vector $A \in V_n$.

An input vector A for which the value 1 is specified for f is said to be a <u>true vector</u> of f. Similarly, an input vector A for which the value of f is specified to be 0 is called a <u>false vector</u> of f. Input vectors of an incompletely specified function f which are neither true nor false vectors (i.e., vectors for which f is unspecified) are said to be <u>unspecified vectors</u> of f.

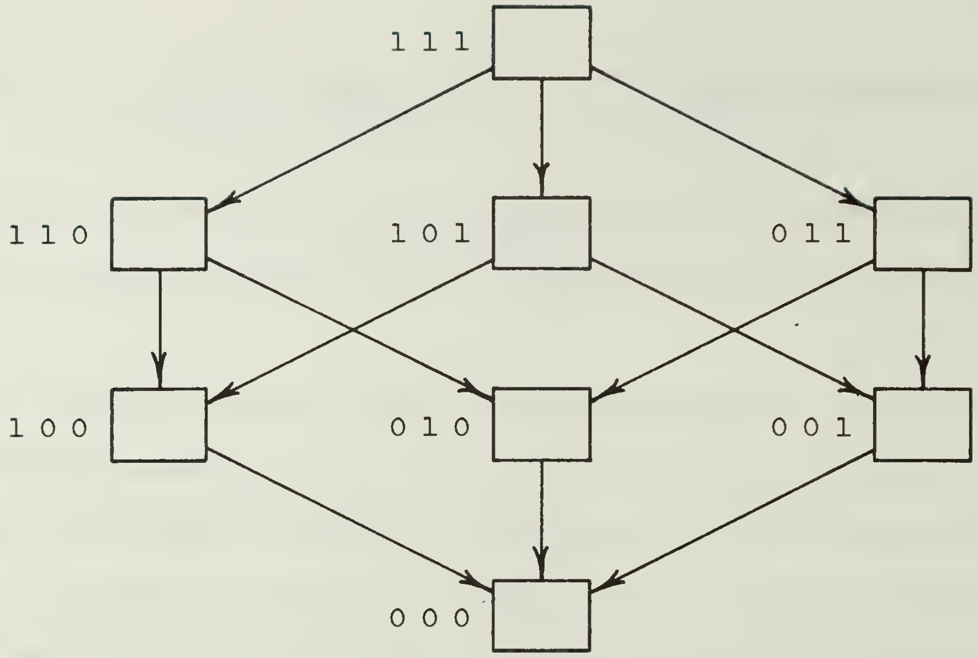


Fig. 2.4 A 3-cube, C_3.

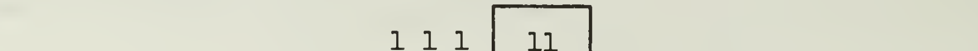


Fig. 2.5 A labelled 3-cube with respect to $f_1 = x_1x_2 \vee x_1x_3 \vee \bar{x}_1\bar{x}_3$ and $f_2 = x_2 \vee x_1\bar{x}_3$.

For any given set, Q, of vectors: vector $A \in Q$ is a *minimum vector* of Q if and only if there exists no vector $B \in Q$ such that $B < A$; $A \in Q$ is a *maximum vector* of Q if and only if there exists no $B \in Q$ such that $B > A$. In particular, if Q is the set of all true vectors of a function f, the maximum vectors of this set are said to be *maximum true vectors* of f. Similarly, if Q is the set of all false vectors of f, the minimum vectors of this set are said to be *minimum false vectors* of f.

Let $f_1,\ldots,f_m$ be completely specified functions of n variables. An n-cube C_n is referred to as a *labelled n-cube* with respect to functions $f_1,\ldots,f_m$, denoted $C_n(f_1,\ldots,f_m)$, when a binary integer,

$$L(A) \equiv \ell(A;f_1,\ldots,f_m) = \sum_{i=1}^{m} f_i(A)\, 2^{m-i},$$

is attached to each vertex A of C_n as a label. Fig. 2.3(c) shows a labelled 3-cube with respect to the function $f = x_1x_2 \vee x_2\bar{x}_3$, and Fig. 2.5 shows a labelled 3-cube with respect to the functions $f_1 = x_1x_2 \vee x_1x_3 \vee \bar{x}_1\bar{x}_3$ and $f_2 = x_2 \vee x_1\bar{x}_3$.

Possibly the best-known characterizations of positive and negative functions are as given in the following two definitions.

Definition 2.1 A (completely specified) Boolean function of n variables, $x_1,\ldots,x_n$, is said to be a *positive function* of these variables if and only if it can be represented by a Boolean expression in which only the non-complemented literals $x_1,\ldots,x_n$ appear.

Definition 2.2 A (completely specified) Boolean function of n variables, $x_1,\ldots,x_n$, is said to be a *negative function* of these variables if and only if it can be represented by a Boolean expression in which only the complemented literals $\overline{x}_1,\ldots,\overline{x}_n$ appear.

Using DeMorgan's law, it is easily seen that a negative function is the complement of a positive function of the same variables.

Based on a theorem in [Gil 54] characterizing positive functions, this similar theorem, appearing as a definition in [IM 71], can be shown for negative functions as defined in Definition 2.2.

Theorem 2.1 A completely specified function, f, of n variables, $x_1,\ldots,x_n$, is a negative function of $x_1,\ldots,x_n$ if and only if for every pair of vectors A and B in V_n such that $A > B$, $f(A) \leq f(B)$.

Other equivalent characterizations of completely specified negative functions are also useful. The following corollary is obvious.

Corollary 2.1 If a completely specified switching function, f, of n variables, $x_1,\ldots,x_n$, is a negative function of these variables and $f(A) = 1$ for some $A \in V_n$, then $f(B) = 1$ for every $B \in V_n$ such that $B < A$. Similarly, if $f(A) = 0$ for some $A \in V_n$, then $f(B) = 0$ for every $B \in V_n$ such that $B > A$.

The following corollary is also obtained from Theorem 2.1.

Corollary 2.2 If for a completely specified switching function, f, of n variables, $x_1,\ldots,x_n$,

$f(B) = 1$ for every $B \in V_n$ for which at least one vector $A \in V_n$ exists such that $B < A$ and $f(A) = 1$ or

$f(B) = 0$ for every $B \in V_n$ for which at least one vector $A \in V_n$ exists such that $B > A$ and $f(A) = 0$,

then f is a negative function of $x_1,\dots,x_n$.

The next theorem is from [IM 71] where it was presented in the form of a theorem concerning columns of a truth table.

Theorem 2.2 A completely specified function f of n variables, $x_1,\dots,x_n$, is a negative function of those variables if and only if for every pair of input vectors, A and B, such that $f(A) = 1$ and $f(B) = 0$, there exists a subscript i $(1 \leq i \leq n)$ such that $a_i = 0$, $b_i = 1$.

Proof First consider the 'if' part of the theorem. Suppose that f is a completely specified function of $x_1,\dots,x_n$ and that for every pair of A and B such that $f(A) = 1$ and $f(B) = 0$, there exists an i such that $a_i = 0$, $b_i = 1$. Since $a_i = 0$ and $b_i = 1$, the relation $A \not> B$ must hold between A and B. Next, it will be proved by contradiction that f must then be a negative function of $x_1,\dots,x_n$. Assume f is not a negative function of $x_1,\dots,x_n$. Then by Theorem 2.1, for at least one pair of A and B, $A > B$ and $f(A) > f(B)$ (equivalently, $f(A) = 1$, $f(B) = 0$). The existence of such a pair contradicts the original premise that $A \not> B$ for every pair A,B such that $f(A) = 1$ and $f(B) = 0$; thus the 'if' part of the theorem must be true.

Next consider the 'only if' part of the theorem. Suppose f is a completely specified negative function of $x_1,\dots,x_n$. It will be proved by contradiction that for every pair of A and B such that $f(A) = 1$ and $f(B) = 0$, there exists an i such that $a_i = 0$, $b_i = 1$. Assume for at least one pair of A and B such that $f(A) = 1$ and $f(B) = 0$, there does not exist such an i. Thus $a_i \geq b_i$ must hold for $i = 1,\dots,n$, implying $A > B$ (A and B are assumed to be distinct). By Theorem 2.1, for such a pair A,B, $f(A) \leq f(B)$, contradicting the premise that $f(A) = 1$ and $f(B) = 0$. Therefore, the 'only if' part of the theorem is also true.

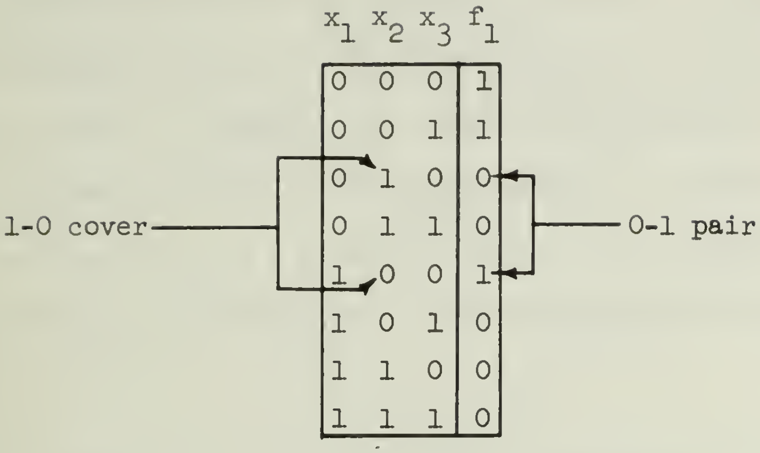
Q.E.D.

This characterization of a completely specified negative function is important in the consideration of truth table representations of negative functions. A combination of a '0' entry and a '1' entry appearing in a column of a truth table for a function f of $x_1, \ldots, x_n$ is referred to as a <u>0-1 pair</u>. A corresponding combination of a '1' entry and a '0' entry in a column for x_i $(1 \leq i \leq n)$ in the same truth table (see Fig. 2.6) is referred to as a <u>1-0 cover</u>. By Theorem 2.2, a completely specified function f of $x_1, \ldots, x_n$ is obviously a negative function of the same variables if and only if every 0-1 pair in a truth table column for f has a 1-0 cover among the columns corresponding to $x_1, \ldots, x_n$. Function f_1 defined by the truth table in Fig. 2.6 can be found to be a negative function of x_1, x_2, x_3 by verifying that a 1-0 cover exists for each of the 15 different 0-1 pairs in the column for f_1. Function f_2 given in the truth table shown in Fig. 2.7 is found not to be a negative function of x_1, x_2, x_3 since the 0-1 pair corresponding to $f(001) = 0$, $f(011) = 1$ has no associated 1-0 cover.

The next definition and following theorem characterize a negative function with respect to its n-cube representation.

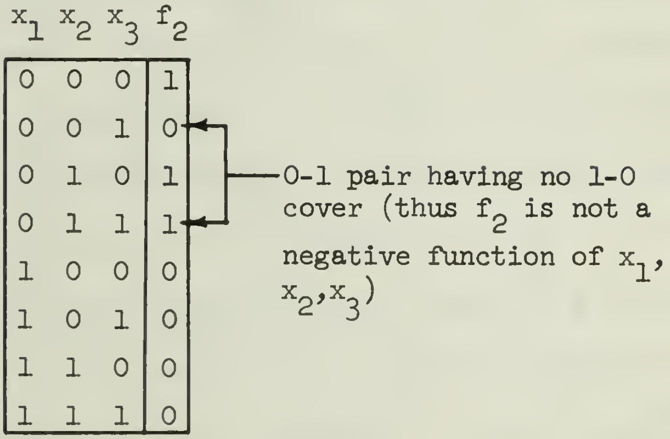
<u>Definition 2.3</u> A directed edge $\overrightarrow{AB}$ of a labelled n-cube $C_n(f_1, \ldots, f_m)$ is said to be an <u>inverse edge</u> if and only if $\ell(A; f_1, \ldots, f_m) > \ell(B; f_1, \ldots, f_m)$.

Figs. 2.8 and 2.9 show $C_n(f_1)$ and $C_n(f_2)$, respectively, where f_1 and



x_1	x_2	x_3	f_1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Fig. 2.6 Example of a 0-1 pair and a corresponding 1-0 cover.



x_1	x_2	x_3	f_2
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

0-1 pair having no 1-0 cover (thus f_2 is not a negative function of x_1, x_2,x_3)

Fig. 2.7 Example of a 0-1 pair having no 1-0 cover.

f_2 are the same two functions of three variables given in the truth tables of Figs. 2.6 and 2.7. In Fig. 2.9, the directed edge $\overrightarrow{(011)(001)}$ shown as a bold line is the only inverse edge in the labelled 3-cube with respect to f_2. The 3-cube labelled with respect to f_1 in Fig. 2.8 has no inverse edges. Fig. 2.5, which shows a 3-cube labelled with respect to two functions, has seven inverse edges (drawn in bold lines).

The following theorem appears in [NTK 72] as the definition of a negative function.

Theorem 2.3 A completely specified switching function, f, of n variables is a negative function of these variables if and only if there is no inverse edge in $C_n(f)$, the labelled n-cube with respect to f.

Proof First, consider the 'if' part of the theorem. Suppose there is no inverse edge in $C_n(f)$. It will be proved by contradiction that f is then a negative function of $x_1,\ldots,x_n$. Assume it is not. Then by Theorem 2.1 there exists at least one pair of vectors A_1 and A_k in V_n (correspondingly, vertices A_1 and A_k in C_n) such that $A_1 > A_k$ and $f(A_1) > f(A_k)$. $f(A_1) > f(A_k)$ implies $f(A_1) = \ell(A_1;f) = 1$ and $f(A_k) = \ell(A_k;f) = 0$. Since $A_1 > A_k$, there must exist a sequence of directed edges $\overrightarrow{A_1A_2}$, $\overrightarrow{A_2A_3}$, ..., $\overrightarrow{A_{k-1}A_k}$ connecting vertices A_1 and A_k in C_n. Since labels with respect to f can only assume values 1 or 0, and since $\ell(A_1,f) = 1$ and $\ell(A_k;f) = 0$, it is obvious that for at least one pair of vertices A_i and A_{i+1}, $1 \leq i < k$, connected by edge $\overrightarrow{A_iA_{i+1}}$, $\ell(A_i;f) = 1$ and $\ell(A_{i+1};f) = 0$. Thus, edge $\overrightarrow{A_iA_{i+1}}$ is an inverse edge, contradicting the original premise. Therefore, f must be a negative function of $x_1,\ldots,x_n$, and the 'if' part of the theorem must hold.

Next, consider the 'only if' part of the theorem. Suppose completely

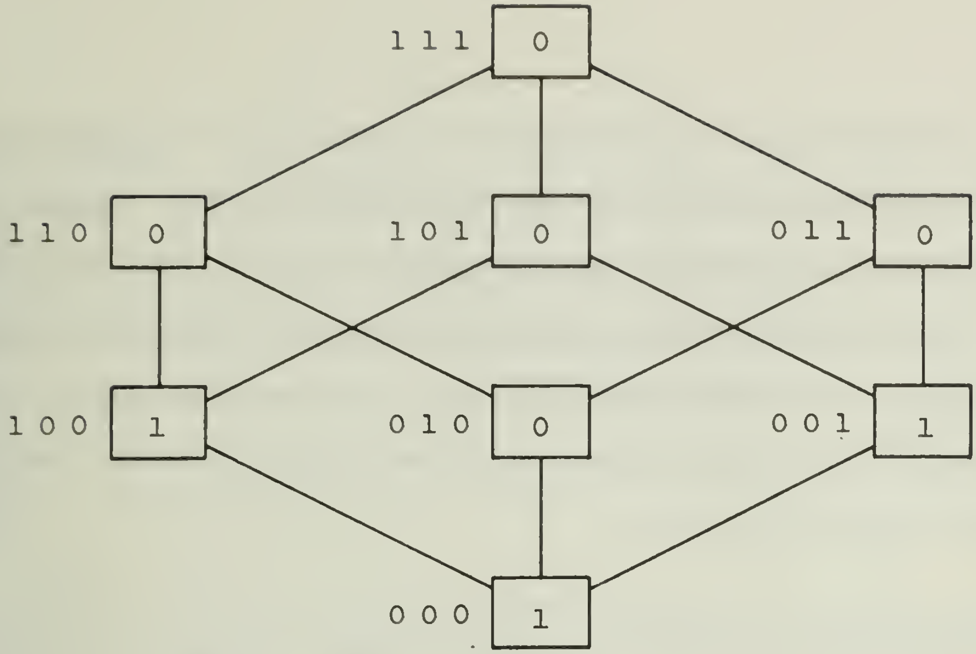


Fig. 2.8 Labelled 3-cube for function f_1 of Fig. 2.6 which has no inverse edge.

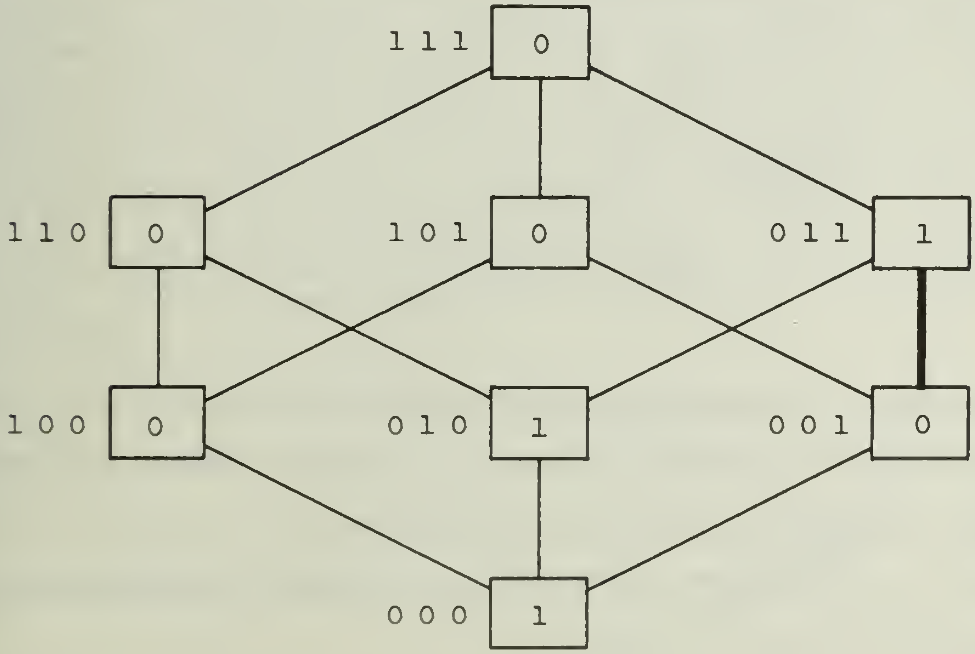


Fig. 2.9 Labelled 3-cube for function f_2 of Fig. 2.7 which has one inverse edge.

specified function f is a negative function of $x_1,\ldots,x_n$. It will be proved by contradiction that there is then no inverse edge in $C_n(f)$. Assume there is such an inverse edge, say $\overrightarrow{AB}$ between vertices A and B. Therefore, $\ell(A;f) > \ell(B;f)$. Since $\ell(A;f) = f(A)$ and $\ell(B;f) = f(B)$, $f(A) > f(B)$. This contradicts Theorem 2.1 which states that no such pair A and B can exist for negative function f. Hence, no inverse edge exists in $C_n(f)$, and the 'only if' part of the theorem must be true.

Q.E.D.

Utilizing Theorem 2.3, it is relatively simple, for small values of n, to check whether or not a completely specified function f is a negative function of n variables by visual inspection of the labelled n-cube $C_n(f)$ (tests for larger values of n are better treated by computer). For example, for n-cubes depicted with vertices of decreasing weights located in increasingly lower physical positions (as in Fig. 2.4), one need only check each edge to see if its 'upper' (i.e., physically higher) vertex has a '1' label while its 'lower' vertex has a '0' label. If such an edge is detected (for example, $\overrightarrow{(011)(001)}$ in Fig. 2.9), it is an inverse edge, and the corresponding function f is not a negative function of $x_1,\ldots,x_n$. Otherwise (for example, see Fig. 2.8), f is a negative function of $x_1,\ldots,x_n$.

<u>Definition 2.4</u> A <u>directed path</u> from a vertex A_1 to a vertex A_q satisfying $A_1 > A_q$ in an n-cube C_n consists of a sequence of directed edges, $\overrightarrow{A_1A_2}$, $\overrightarrow{A_2A_3}$, ..., $\overrightarrow{A_{q-1}A_q}$, connecting the two vertices.

In Fig. 2.4, $\overrightarrow{(111)(110)}$, $\overrightarrow{(110)(010)}$ and $\overrightarrow{(111)(011)}$, $\overrightarrow{(011)(010)}$ are two directed paths from (111) to (010).

Definition 2.5 In a labelled n-cube, $C_n(f)$, for a function f of n variables, the number of inverse edges included in a directed path, p, between two vertices A and B, $A > B$, is called the number of inversions of path p. The *inversion degree of a vertex A with respect to a vertex B* in $C_n(f)$ is the maximum number of inversions over all directed paths from vertex A to vertex B, is defined only for A and B satisfying $A > B$, and is denoted $D^f(A,B)$.

In Fig. 2.10, $D^f((0111),(0001)) = 1$, $D^f(\vec{1},(0001)) = 2$ (inverse edges are shown in bold lines).

Definition 2.6 The *inversion degree of a function f* is the maximum number of inversions over all directed paths from vertex $\vec{1}$ to vertex $\vec{0}$ in $C_n(f)$, i.e., $D^f(\vec{1},\vec{0})$.

The function f shown in Fig. 2.10 in the form of a labelled 4-cube is of inversion degree two since $D^f(\vec{1},\vec{0}) = 2$. It is apparent that the inversion degree of a function of four variables can be no greater than two.

It has been shown that the inversion degree of a function is related to the minimum number of MOS cells required to realize it. This will be discussed later.

Much of the preceding discussion dealt with completely specified negative functions. Next, incompletely specified negative functions will be defined and characterizations corresponding to those for completely specified negative functions given.

Definition 2.7 A *completion* of an incompletely specified function f

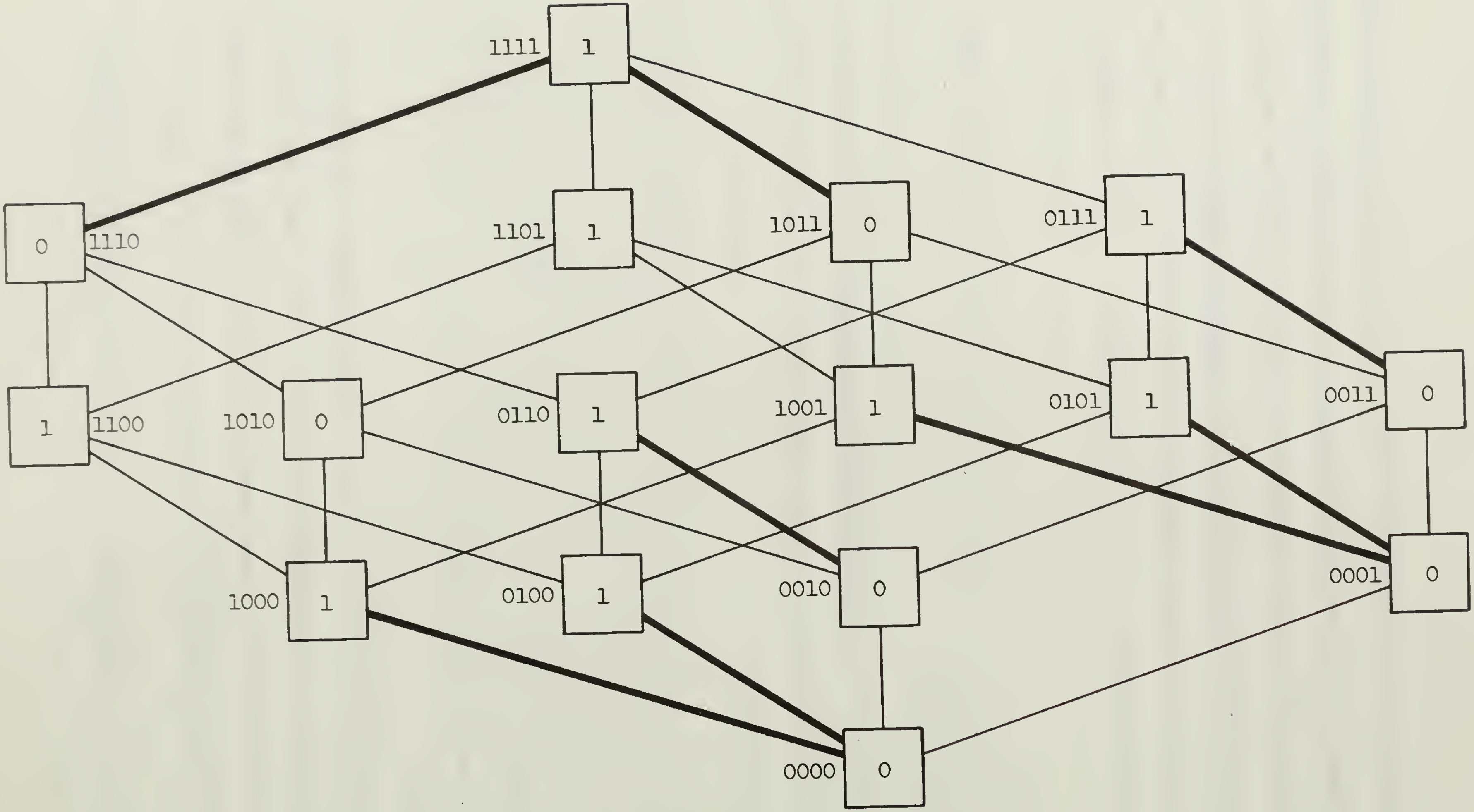


Fig. 2.10 Labelled 4-cube for a function f of inversion degree two.

of n variables, $x_1,\ldots,x_n$, is a completely specified function obtained from f by specifying each unspecified value of f to be either '1' or '0'. The resulting completion is called a negative completion of f if and only if it is a negative function of $x_1,\ldots,x_n$.

Definition 2.8 Incompletely specified function f of $x_1,\ldots,x_n$ is said to be negative with respect to $x_1,\ldots,x_n$ if and only if a negative completion of f exists.

For convenience, a function which is negative with respect to $x_1,\ldots,$ x_n may sometimes be referred to as an incompletely specified negative function (of $x_1,\ldots,x_n$).

For an incompletely specified function, some of its values are unspecified for certain input vectors. Such an unspecified value is denoted '*'.

A characterization of an incompletely specified function of n variables which is negative with respect to those variables is given in the next theorem. This theorem, a composite of two theorems appearing in [IM 71], corresponds to Theorem 2.2 for completely specified functions. The proof of the theorem, also appearing in [IM 71], is repeated here as it contains an algorithm to obtain one negative completion of a given incompletely specified negative function.

Theorem 2.4 An incompletely specified function, f, of n variables, $x_1,\ldots,x_n$, is negative with respect to $x_1,\ldots,x_n$ if and only if for each pair of input vectors A and B for which values of f are specified and f(A) = 0 and f(B) = 1, there exists a subscript i ($1 \leq i \leq n$) such that $a_i = 1$ and $b_i = 0$.

Proof The 'only if' part of the theorem is easily demonstrated. Suppose f is negative with respect to $x_1,\ldots,x_n$, but assume there exists a pair of input vectors, A,B, for which f(A) = 0 and f(B) = 1 are specified and no i exists such that $a_i = 1$ and $b_i = 0$. Then no completion of f can be a negative function of $x_1,\ldots,x_n$ by Theorem 2.2, and, consequently, f can not be negative with respect to $x_1,\ldots,x_n$, contradicting the original premise. Thus, the 'only if' part of the theorem must be valid.

Next consider the 'if' part of the theorem. It is sufficient to show that a negative completion of f always exists given the fact that there exists an i such that $a_i = 1$, $b_i = 0$ for each specified pair of values f(A) = 0, f(B) = 1.

Consider the completion, f', of f obtained as follows ($A,B \in V_n$):

(1) Set f'(A) = 0 for every A such that f(A) = 0;

(2) Set f'(A) = 0 for every A such that f(A) = * and there exists a a B for which A > B and f(B) = 0;

(3) Set f'(A) = 1 for every A not satisfying rule (1) or (2).

It is obvious that f' is a completion of f. Now f' must be proved to be a negative function of $x_1,\ldots,x_n$. The proof will be by contradiction:

Suppose f' is not a negative function of $x_1,\ldots,x_n$. Then by Theorem 2.3, there must exist at least one inverse edge, say $\overrightarrow{BA}$, in $C_n(f')$. There are the following four possible cases since f'(B) = 1 and f'(A) = 0:

Case 1 f(B) = 1 and f(A) = 0.

Since $\overrightarrow{BA}$ is an edge directed from B to A, B > A, contradicting the assumption that there exists an i such that $a_i > b_i$.

Case 2 f(B) = * and f(A) = 0.

By rule (2) above, $f'(B) = 0$, $f'(A) = 0$, and this contradicts the assumption that $\overrightarrow{BA}$ is an inverse edge in $C_n(f')$.

Case 3 $f(B) = 1$ and $f(A) = *$.

Since $\overrightarrow{BA}$ is an inverse edge in $C_n(f')$, $f'(A) = 0$ must hold. $f'(A) = 0$ can only be specified by rule (2) above. Hence there must exist a $D \in V_n$ such that $D < A$ and $f(D) = 0$. Considering vectors D and B, $D < A < B$, $f(B) = 1$, and $f(D) = 0$. However, $D < B$ implies $d_i \leq b_i$, $i = 1,\ldots,n$, and this contradicts the assumption that there exists an i such that $d_i > b_i$.

Case 4 $f(B) = *$ and $f(A) = *$.

Since $\overrightarrow{BA}$ is an inverse edge in $C_n(f')$, $f'(A) = 0$ must hold and can only result from rule (2) above. Thus there exists a $D \in V_n$ such that $D < A$ and $f(D) = 0$. Since $D < A < B$, rule (2) also assigns $f'(B) = 0$. This contradicts the assumption that $\overrightarrow{BA}$ is an inverse edge in $C_n(f')$.

This proves the 'if' part of the theorem.

Q.E.D.

The next two theorems, appearing in [Lai 76], extend the results of Corollary 2.1 and 2.2 to the case of incompletely specified functions. These characterizations of incompletely specified negative functions are often useful in determining whether or not a given function — either completely or incompletely specified — is negative.

Theorem 2.5 A function f of n variables is negative with respect to these variables if and only if for every vector $A \in V_n$ such that $f(A) = 0$, either $f(B) = 0$ or $f(B) = *$ for every vector $B \in V_n$ satisfying $B > A$.

Proof First consider the 'if' part of the theorem. Suppose for every A such that $f(A) = 0$, $f(B) = 0$ or $*$ for every B such that $B > A$. Now consider any pair of vectors $A,B \in V_n$ such that $f(A) = 0$ and $f(B) = 1$. Then, by the original premise, $B \not> A$ must hold. Thus, either B is incomparable to A or $B < A$. In either case there must exist an i such that $a_i > b_i$. By Theorem 2.4 f must then be a negative function with respect to $x_1,\ldots,x_n$, and the 'if' part of the theorem is true.

Now consider the 'only if' part of the theorem. Suppose f is negative with respect to $x_1,\ldots,x_n$. The proof will be by contradiction. Assume there exists at least one $A \in V_n$ such that $f(A) = 0$ and at least one $B \in V_n$ such that $B > A$ and $f(B) = 1$. Then for every completion, f', of f, $f'(A) = 0$ and $f'(B) = 1$. By Theorem 2.1, no such f' can be a negative function of $x_1,\ldots,x_n$. The resulting implication that f is not negative with respect to $x_1,\ldots,x_n$ contradicts the original premise. Hence the 'only if' part of the theorem holds.

Q.E.D.

Theorem 2.6 A function f of n variables is negative with respect to these variables if and only if for every vector $A \in V_n$ such that $f(A) = 1$, either $f(B) = 1$ or $f(B) = *$ for every vector $B \in V_n$ satisfying $B > A$.

Proof Proof is similar to that for Theorem 2.5.

Corresponding to characterizations previously given for completely specified functions, further characterizations of incompletely specified negative functions can be presented. The following corollary, corresponding to Theorem 2.1 for completely specified functions, is obvious from the two preceding theorems.

Corollary 2.3 An incompletely specified function, f, of n variables is negative with respect to these variables if and only if $f(A) \leq f(B)$ holds for every pair of vectors $A,B \in V_n$, $A > B$, for which values of f are specified.

From the fact that the driver of an MOS cell operates as a two-terminal network of normally-open relay contacts, it is easily seen that the transmission function of such a network must be a positive function of its inputs. Furthermore, it is clear that any positive function of n variables, $x_1,\ldots,$ x_n, can (theoretically) be realized by the driver of an MOS cell (obviously, a Boolean expression for f or its completion, in terms of the non-complemented variables $x_1,\ldots,x_n$ only, must exist and can be implemented by a two-terminal FET network in a straightforward manner).

As pointed out earlier, if the driver of an MOS cell realizes function f, the MOS cell itself realizes $\bar{f}$. Hence, by DeMorgan's law, $\bar{f}$ is a negative function of n variables if and only if f is a positive function of the same variables. Based upon this discussion, the following theorem is obvious.

Theorem 2.6 The output function of an MOS cell is always a negative function with respect to the inputs of the cell. A negative function of n variables, $x_1,\ldots,x_n$, can always be realized by an MOS cell with inputs $x_1,\ldots,x_n$.

While Theorem 2.6 is valid from a theoretical viewpoint, it must be noted that it is actually impractical to construct MOS cells consisting of more than a certain number of MOSFET's. This number, however, is not fixed and depends on such things as the operational mode (i.e., static, dynamic, or complementary) and the required speed.

Obviously, any Boolean function f can be expressed as a negative function of $x_1,\ldots,x_k,x_{k+1},\ldots,x_{2k}$ where $x_{k+j} = \bar{x}_j$, $j = 1,\ldots,k$. Thus, any function f can be realized by a single MOS cell if both variables and their complements are available as inputs.

This paper will assume complements of variables are not available. The functions available to a network of MOS cells as inputs are n independent variables $x_1,\ldots,x_n$ which will be referred to as external variables. Let X denote the set of external variables $\{x_1,\ldots,x_n\}$.

Under the restriction that only non-complemented variables are available as inputs, determining the number of cells required to realize a given Boolean function is not a trivial problem. There is, however, an easily derived upper bound on the number of cells necessary to realize a given function. Since the complement, $\bar{x}_i$, of each external variable, x_i, can be obtained using a single MOS cell as a simple inverter, the complements of all inputs can be realized with n cells. With the resulting availability of all variables and complements of variables, f can be realized with one additional cell — a total of n+1 cells in the network. In general, however, networks of fewer MOS cells can be found.

The theoretical model for an MOS cell is the negative gate. A negative gate is assumed to be capable of realizing any negative function of its inputs. Once the desired function has been determined, an MOS cell of a specific structure can be substituted for the negative gate. Corresponding to networks of MOS cells are networks of negative gates (also referred to as negative gate networks). The generalized form of a feed-forward network of negative gates is shown in Fig. 2.11: let g_i denote the i-th gate

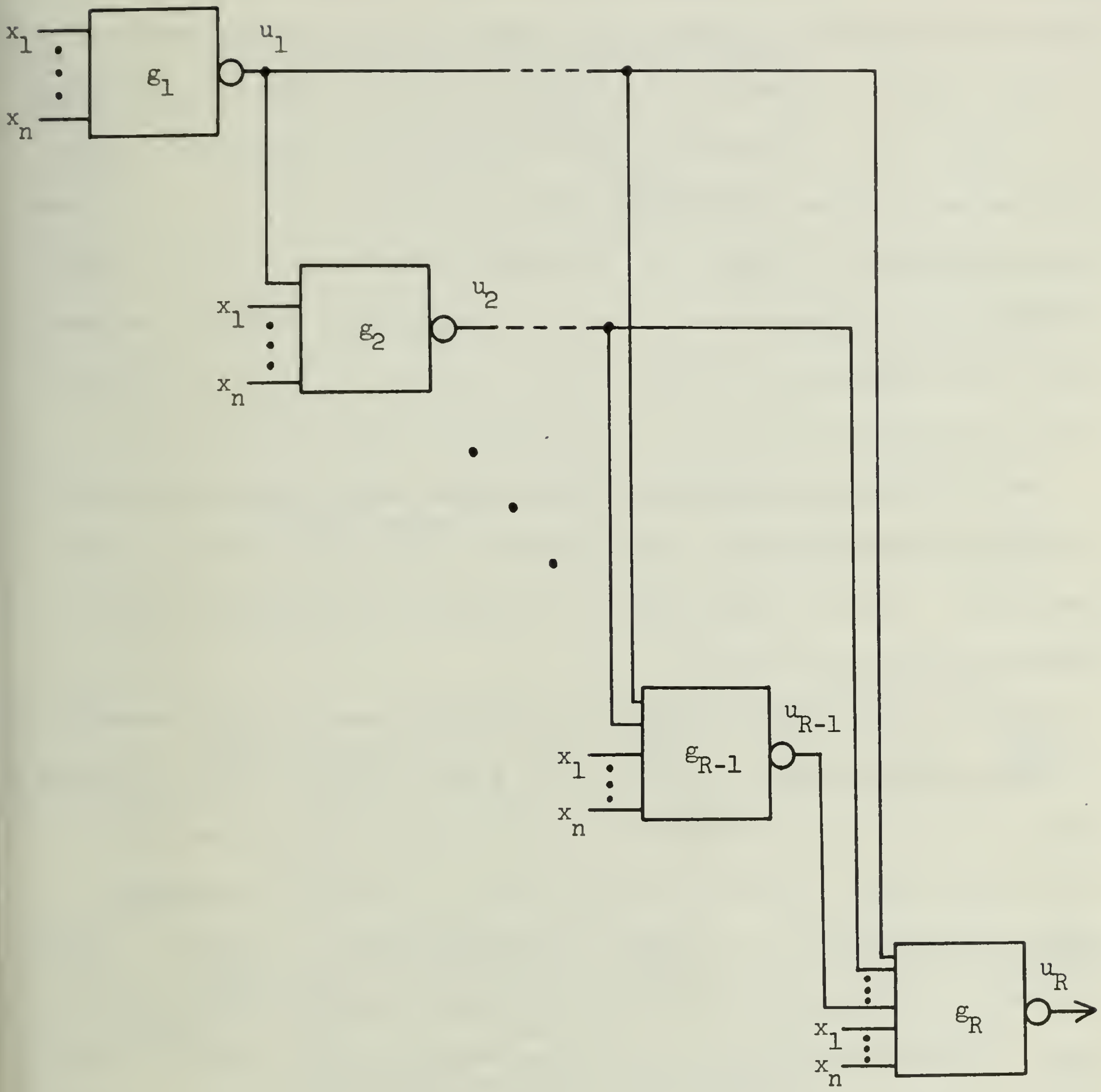


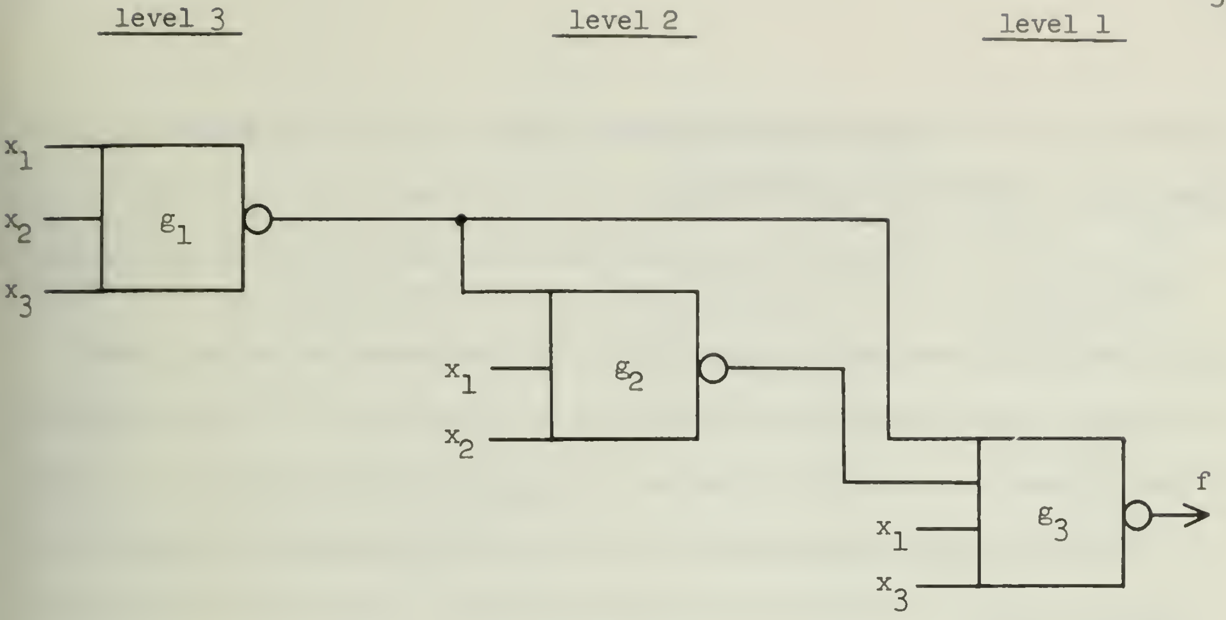
Fig. 2.11 Generalized form of a feed-forward network of R negative gates.

from the left, $i = 1,\ldots,R$, and let $u_i(x_1,\ldots,x_n)$ denote the completely specified function realized by gate g_i with respect to the external variables $x_1,\ldots,x_n$. Since each gate g_i is a negative gate with respect to its inputs, $x_1,\ldots,x_n,g_1,\ldots,g_{i-1}$, function u_i can be considered an incompletely specified function of these n+i-1 variables: $u_i(x_1,\ldots,x_n,u_1,\ldots,u_{i-1})$. All feed-forward negative gate networks can be obtained from Fig. 2.11 by the deletion of some (or no) connections. The form in Fig. 2.11 maintains its generality since variables in functions $u_i(x_1,\ldots,x_n,u_1,\ldots,u_{i-1})$ corresponding to deleted connections can be considered to be dummy variables of u_i.

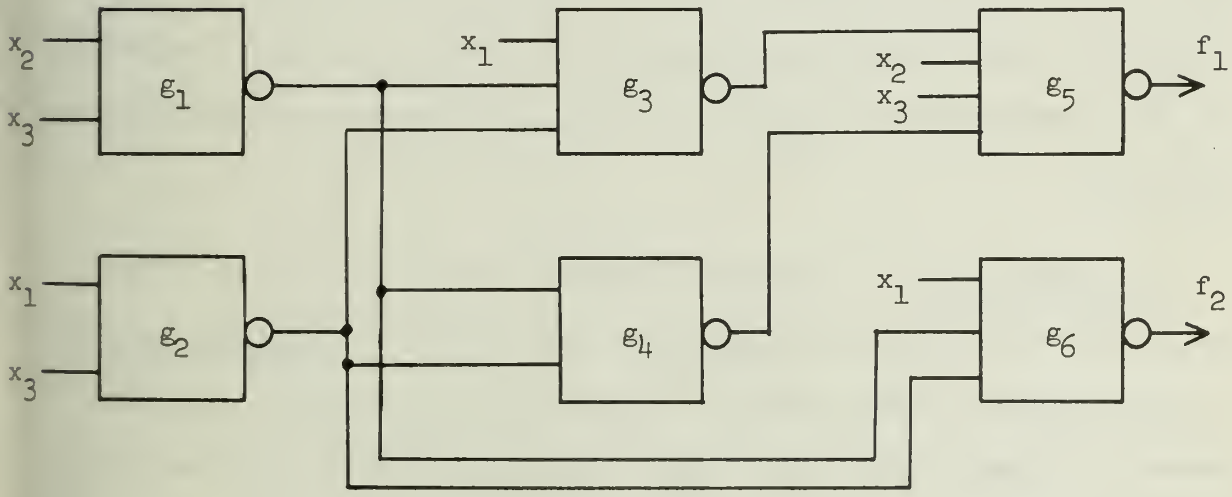
Terms previously defined for networks of MOS cells are also applicable to networks of negative gates (e.g., immediate predecessor, external variables, etc.)。 Further terminology is now defined for both negative gate networks and MOS cell networks.

Algorithms given later synthesize MOS cell or negative gate networks to realize either a single function f or a group of functions $f_1,\ldots,f_m$ (the letter m is used exclusively for denoting the number of functions to be realized by a single network). Such an f or f_i is said to be an _output function of the network_ (or a _network output function_). In a network, each gate (cell) realizing a network output function is called an _output gate (cell)_. Networks realizing a single output function are distinguished from those realizing, simultaneously, two or more output functions by referring to the former as _single-output networks_ and to the latter as _multiple-output networks_.

Examples of single- and multiple-output negative gate networks appear in Figs. 2.12(a) and 2.12(b), respectively. Fig. 2.2 shows a multiple-output



(a) Example of a single-output negative gate network of three levels.



(b) Example of a multiple-output negative gate network of three levels.

Fig. 2.12 Single- and multiple-output negative gate networks.

network of MOS cells (the output functions being $f_1 = S_i = A_i \oplus B_i \oplus C_{i-1}$ and $f_2 = \overline{C}_i = \overline{A_iB_i \vee A_iC_{i-1} \vee B_iC_{i-1}}$).

Definition 2.9 The level of a negative gate (MOS cell), g_i, in a network is defined as the maximum number of negative gates (MOS cells) on any path from g_i to any output gate (MOS cell) of the network.[†] A negative gate (MOS cell) whose level is ℓ is said to be in level ℓ of the network. A network of negative gates (MOS cells) is said to be a network of ℓ levels (or an ℓ-level network) if and only if the highest level of any gate (MOS cell) in the network is ℓ.

By this definition, in the network of Fig. 2.12(b), gates g_5 and g_6 are in level 1, gates g_3 and g_4 are in level 2, and gates g_1 and g_2 are in level 3 of the network; and the network itself is thus a 3-level network. In Fig. 2.2, MOS cells g_1, g_2, and g_3 are in levels 2, 1, and 1, respectively; the network made up of these three cells is therefore a network of only two levels.

Definition 2.10 For networks of negative gates and networks of MOS cells, the following types of optimality are defined: A G-minimal network is one having the minimum number of negative gates (MOS cells) among all networks of negative gates (MOS cells) realizing switching function f (or group of functions $f_1,\ldots,f_m$). An I-minimal network is one having the minimum number of intercell (intergate) connections among all networks realizing

[†] Although levels are commonly numbered in the reverse order (input gates to output gates), this definition is convenient for later algorithms and computer program implementations which consider a network's gates while generally moving from the output gates to the input gates.

switching function f (or $f_1,\ldots,f_m$). An F-minimal network of MOS cells is one having the minimum number of FET's (both driver and load FET's included) among all MOS cell networks realizing switching function f (or $f_1,\ldots,f_m$).

Combinations of the letters G, I, and F can also be used to represent the combination of two or three of the above optimality criteria. For example, a GI-minimal network of negative gates for a function f is one having the minimum number of intergate connections among all those networks having the minimum number of gates among all negative gate networks realizing f. In general: the left-most letter will designate the optimality criterion of primary importance; the second letter will designate the criterion of secondary importance; and the third letter (if any) will designate the criterion of tertiary importance.

3. PROPERTIES RELATED TO OPTIMAL NETWORKS

This section will present a collection of properties related to optimal negative gate and MOS cell networks. Some of the properties deal exclusively with G-minimal networks, and others, important to the reduction of series-parallel problems with driver FET's in both optimal and non-optimal MOS cell networks, relate indirectly to F-minimal networks.

Some of the properties, including the two most important concerning the number of negative gates in two-level-restricted and non-level-restricted G-minimal networks, have appeared elsewhere.[IM 71][Iba 71][NTK 72][Liu 72]

The properties in themselves are insufficient to provide a method for synthesizing optimal networks of various types. However, they are useful for: checking designs for optimality; testing for the lack of necessary characteristics of optimal networks in a design or partial design (perhaps, as part of a heuristic or algorithmic synthesis process); and characterizing, at least partially, optimal negative gate and (both optimal and non-optimal) MOS cell networks.

The following theorem is demonstrated in [Liu 72].

Theorem 3.1 In a single-output, G-minimal, negative gate (MOS cell) network, each gate (cell) other than the output gate (cell) must have at least one input from the set of external variables.

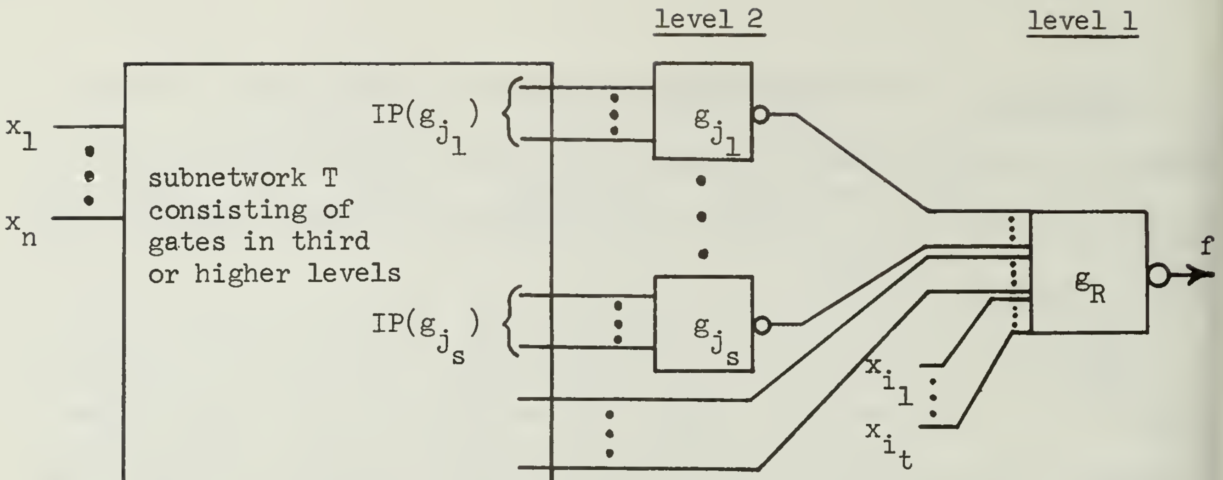
The next group of new theorems and corollaries relate certain sets of inputs to the numbers of negative gates (MOS cells) appearing in the second or third levels of a G-minimal network.

Theorem 3.2 In a single-output, G-minimal, negative gate (MOS cell) network, the number of gates (cells) in the second level of the network is at most $\lceil \log_2(t + 1)\rceil + 1$[†] where t is the number of external variable inputs to the output gate (cell).

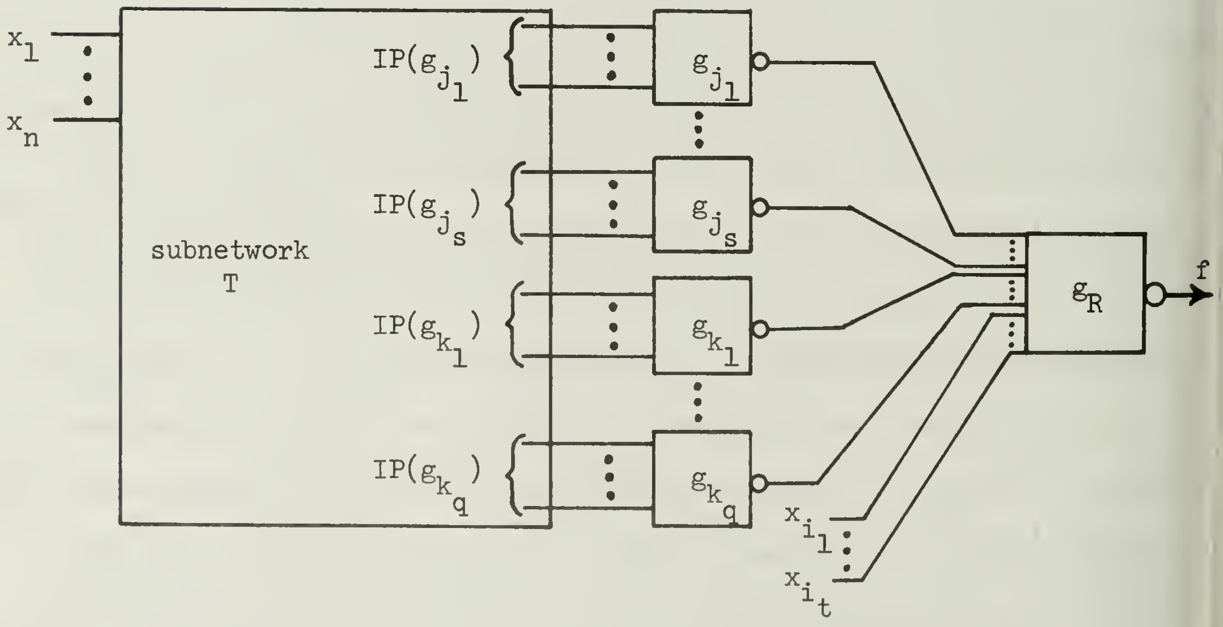
Proof This theorem can be proved by demonstrating that a contradiction occurs if it is assumed to be false. Assume there exists a single-output, G-minimal, negative gate network (the use of MOS cells does not affect the proof) for which: the output gate, g_R, has t external variable inputs, x_{i_1}, $\ldots, x_{i_t}$; the second level consists of s gates, $g_{j_1}, \ldots, g_{j_s}$; and $s > \lceil \log_2 (t + 1)\rceil + 1$. (Such a network is illustrated in Fig. 3.1(a).)

If there exist additional inputs to g_R from gates in level three or higher, duplicate all such gates and use the new duplicates, $g_{k_1}, \ldots, g_{k_q}$, of the gates as inputs to g_R in place of the respective originals. This results in a second level of s + q gates (see Fig. 3.1(b)). Next add two inverters to the network in series with the output of g_R. Since an inverter is a special case of a negative gate, this adds two more negative gates, g_A and g_B, realizing $\overline{f}$ and f, respectively, to the network (see Fig. 3.1(c)). Now, since it is apparent (refer to Fig. 3.1(c)) that f realized by g_R is a positive function of $IP(g_{j_1}) \cup \ldots \cup IP(g_{j_s}) \cup IP(g_{k_1}) \cup \ldots \cup IP(g_{k_q}) \cup \{\overline{x}_{i_1}, \ldots, \overline{x}_{i_t}\}$, $\overline{f}$ realized by g_A must be a negative function of these same variables. Create a new negative gate, g_A', to realize this negative function and provide one new negative gate to obtain the necessary complement of each of $x_{i_1}, \ldots, x_{i_t}$ (see Fig. 3.1(d)). Actually, it is not necessary to use t negative gates to obtain $\overline{x}_{i_1}, \ldots, \overline{x}_{i_t}$ from $x_{i_1}, \ldots, x_{i_t}$. Due to

[†] $\lceil p\rceil$ for a real number p denotes the smallest integer not smaller than p.

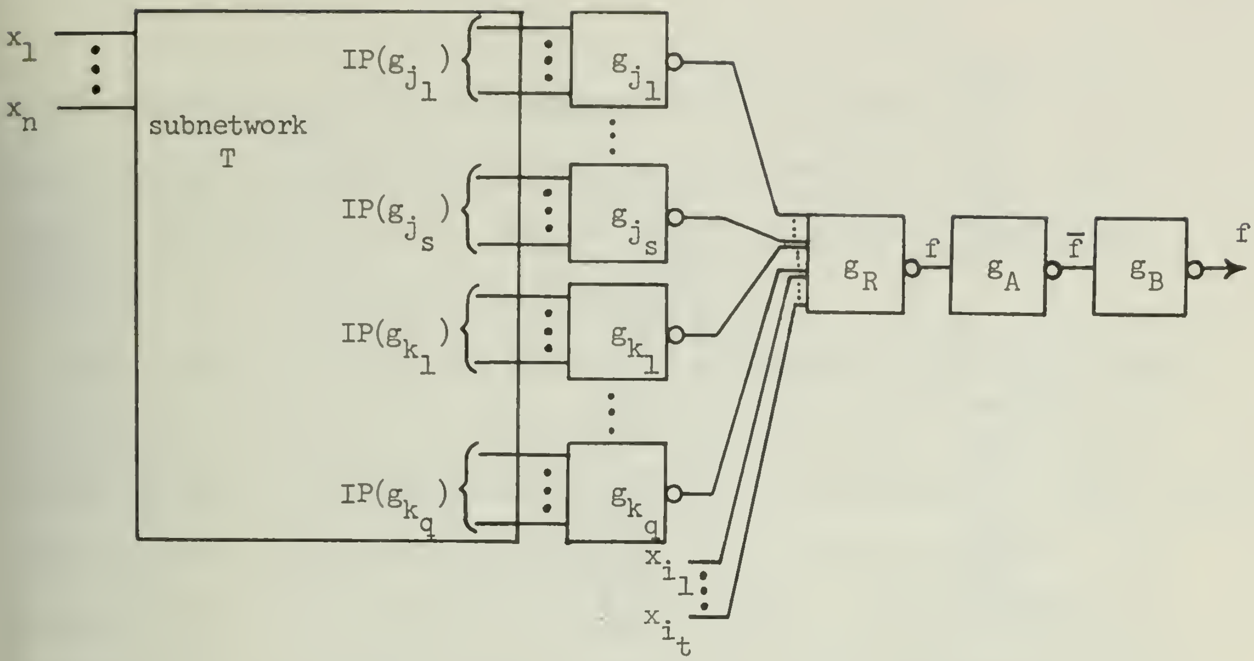


(a) Network realizing f. Network is assumed to be optimal with $s > \lceil \log_2(t + 1) \rceil + 1$.

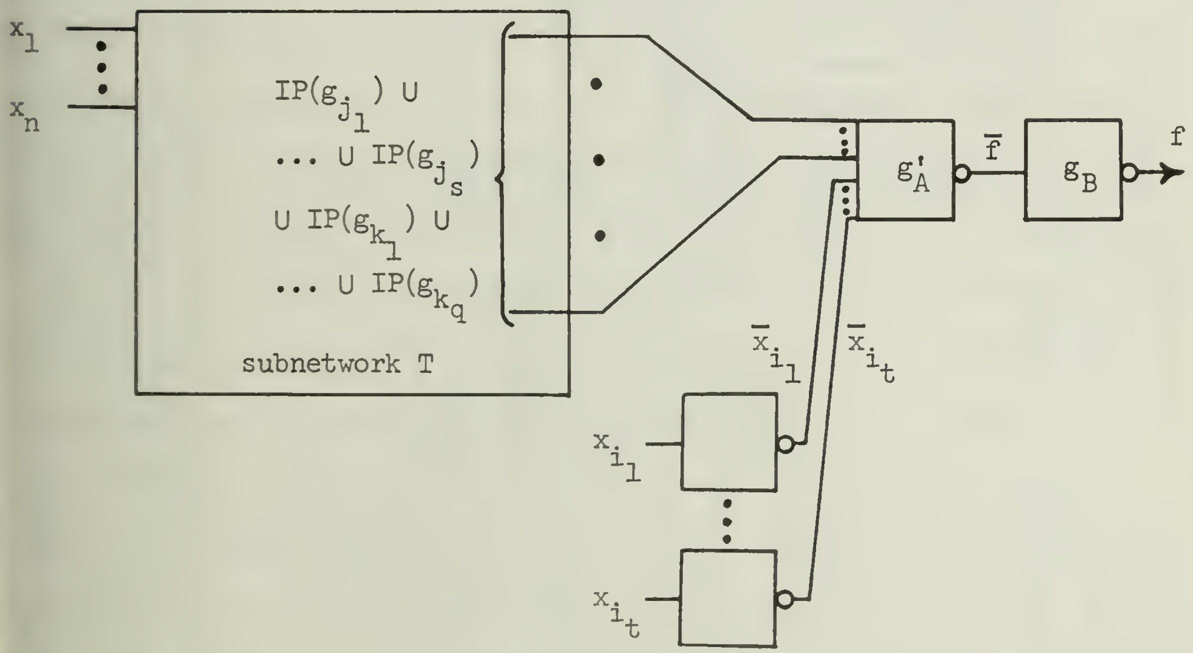


(b) Network after duplication of every negative gate in level three or higher which was an input of g_R.

Fig. 3.1 Illustration of the proof for Theorem 3.1.

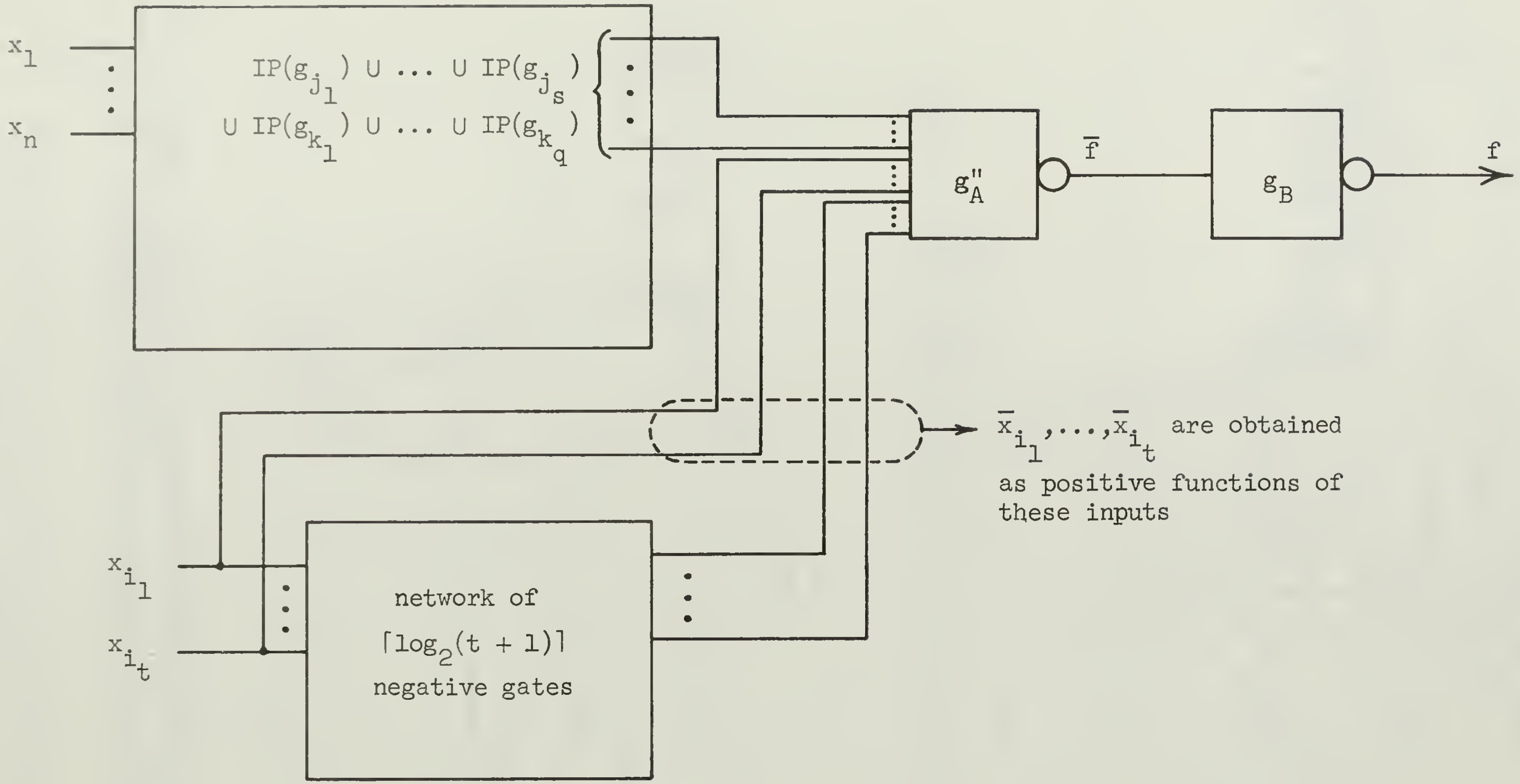


(c) Network after adding two negative gates, each acting as a simple inverter, following the output gate.



(d) Network after replacement of gates $g_{j_1},\ldots,g_{j_s},g_{k_1},\ldots,g_{k_q},g_R$, and g_A by a new gate g'_A and gates realizing the complements of $x_{i_1},\ldots,x_{i_t}$.

Fig. 3.1 (Continued)


x_1
x_n
$IP(g_{j_1}) \cup \ldots \cup IP(g_{j_s})$
$\cup\ IP(g_{k_1}) \cup \ldots \cup IP(g_{k_q})$
g''_A
$\overline{f}$
g_B
f
$\overline{x}_{i_1},\ldots,\overline{x}_{i_t}$ are obtained as positive functions of these inputs
x_{i_1}
x_{i_t}
network of $\lceil \log_2(t+1) \rceil$ negative gates

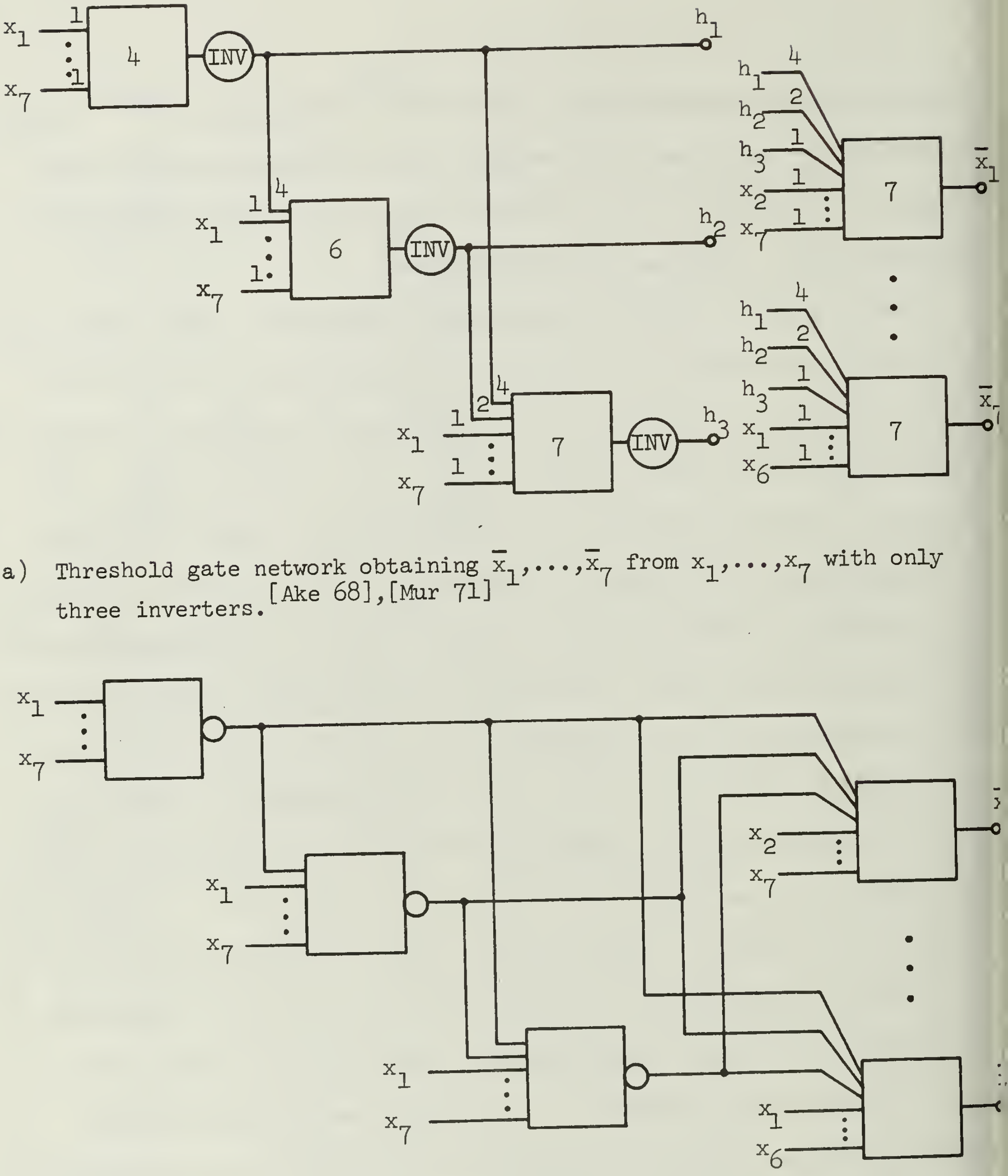
(e) Final network in which g'_A and the t negative gates realizing the complements of $x_{i_1},\ldots,x_{i_t}$ are replaced by a new gate g''_A, also realizing $\overline{f}$, and $\lceil \log_2(t+1) \rceil$ additional negative gates.

Fig. 3.1 (Continued)

results obtained independently by Markov[Mar 58] and Muller,[Mul 58] it is known that the complements of p variables can be obtained using at most $\lceil \log_2(p + 1)\rceil$ inverters, i.e., at most $\lceil \log_2(p + 1)\rceil$ negative gates plus additional gates realizing positive functions.

Their proofs are abstract, but simpler constructive proofs are presented in [Ake 68] and [Mur 71]. By these constructions, $\bar{x}_{i_1},\ldots,\bar{x}_{i_t}$ can be obtained as positive functions of $x_{i_1},\ldots,x_{i_t}$ and the outputs of $\lceil \log_2(t + 1)\rceil$ negative gates (with inputs only from $x_{i_1},\ldots,x_{i_t}$, and each other). Furthermore, a network so obtained is always loop-free. (Fig. 3.2(a) shows an example from [Ake 68],[Mur 71] in which $\bar{x}_1,\ldots,\bar{x}_7$ are obtained from x_1, $\ldots,x_7$ using only three inverters. The gate used are threshold gates with only positive input weights. Since such threshold gates obviously realize positive functions, the corresponding network of three negative gates and seven positive gates shown in Fig. 3.2(b) can be obtained.) Thus $\bar{f}$, currently realized by g'_A, is a negative function of these $\lceil \log_2(t + 1)\rceil$ negative gates, external variables $x_{i_1},\ldots,x_{i_t}$, and $IP(g_{j_1}) \cup \ldots \cup IP(g_{j_s}) \cup IP(g_{k_1}) \cup \ldots \cup IP(g_{k_q})$. The result is a network realizing function f (see Fig. 3.1(e)) in which the s + 1 gates constituting levels one and two of the original network have been replaced by $\lceil \log_2(t + 1)\rceil + 2$ gates (the number of gates in the remainder of the network has not been changed). Since the original network was supposed to be G-minimal, the final network must have at least as many gates. Thus, $\lceil \log_2(t + 1)\rceil + 2 \geq s + 1$ (i.e., $\lceil \log_2(t + 1)\rceil + 1 \geq s$) must hold. However, this contradicts the assumption that $s > \lceil \log_2(t + 1)\rceil + 1$. Therefore, the theorem must be true.

Q.E.D.



(a) Threshold gate network obtaining $\overline{x}_1,\ldots,\overline{x}_7$ from $x_1,\ldots,x_7$ with only three inverters. [Ake 68],[Mur 71]

(b) Equivalent network of negative and positive gates obtaining $\overline{x}_1,\ldots,\overline{x}_7$ from $x_1,\ldots,x_7$ with only three negative gates.

Fig. 3.2 Feedforward networks inverting t (= 7) variables with only $\lceil \log_2(t + 1) \rceil$ inverters.

Observing that the proof of Theorem 3.2 can be duplicated with respect to gates other than an output gate, the theorem can be restated in a form applicable to all gates of a G-minimal negative gate network:

Theorem 3.3 For every gate (cell) g_i in a single-output, G-minimal, negative gate (MOS cell) network: the number of gates (cells) having only one output connection apiece which are inputs of g_i (g_i may have inputs from other gates (cells) as well) is at most $\lceil \log_2(t + 1)\rceil + 1$, where t is the number of external variable inputs to g_i.

These results can be extended to apply to a group of gates rather than just a single gate, and the restriction on the number of gates with single output connections which can be inputs to the same gate can also be tightened in certain circumstances.

Corollary 3.1 In a single-output, G-minimal, negative gate (MOS cell) network having an output gate (cell) with no external variable inputs, no more than one gate exists in the second level of the network.

For a negative gate (MOS cell) with up to ten external variable inputs in a single-output, G-minimal network, Table 3.1 shows values corresponding to the limits given by Theorem 3.3.

Theorem 3.4 In a single-output, G-minimal, negative gate (MOS cell) network, if the number of gates (cells) in the second level equals $\lceil \log_2 (t + 1)\rceil + 1$, where t is the number of external variable inputs to the output gate (cell), then there exists a G-minimal negative gate (MOS cell)

Number of external variable inputs to a gate g_i: (t =)	Maximum number of gates with single output connections which are inputs of g_i: ($\lceil \log_2(t + 1)\rceil + 1$ =)
0	1
1	2
2	3
3	3
4	4
5	4
6	4
7	4
8	5
9	5
10	5

Table 3.1 Relations among inputs to a gate g_i in a single-output, G-minimal, negative gate network.

Number of external variables in set IP(IP(g_R)): (r =)	Maximum number of second and third level gates in the network: ($\lceil \log_2(r + 1)\rceil$ =)
0	0
1	1
2	2
3	2
4	3
5	3
6	3
7	3
8	4
9	4
10	4

Table 3.2 Relations between the number of second and third level gates and the number of external variables in IP(IP(g_R)) for a single-output, G-minimal, negative gate network.

network, realizing the same output function, in which the output gate (cell) performs the function of a simple inverter.

Proof If the output gate of the given network does not already perform the inversion function, consider the procedure given in the proof of Theorem 3.2. Let w be the number of gates in subnetwork T (see Fig. 3.1(a)) of the network in this procedure. Then this procedure produces from a G-minimal network of $w + s + 1$ gates, a network of $w + \lceil \log_2(t + 1)\rceil + 2$ gates having a simple inverter as its output gate. If the number of gates in the second level of the original network equals $\lceil \log_2(t + 1)\rceil + 1$ (i.e., $s = \lceil \log_2(t + 1)\rceil + 1$), then both the original and final networks have the same number of gates. Therefore, the network produced by the procedure is also G-minimal.

Q.E.D.

When the conditions of this theorem are met, it may be useful in designing several MOS cell (negative gate) networks which are to be interconnected. An output cell configuration which is a simple inverter means the complement, $\overline{f}$, of the realized function f requires one less cell. If f is not really needed in explicit form (e.g., if it was just chosen as an intermediate function during the partitioning of a large section of logic), $\overline{f}$ can sometimes serve just as well as an input to subsequent networks. (As an example, see the 1-bit adder in Fig. 2.2 in which the carry is not realized explicitly, but implicitly as the complement of the carry.)

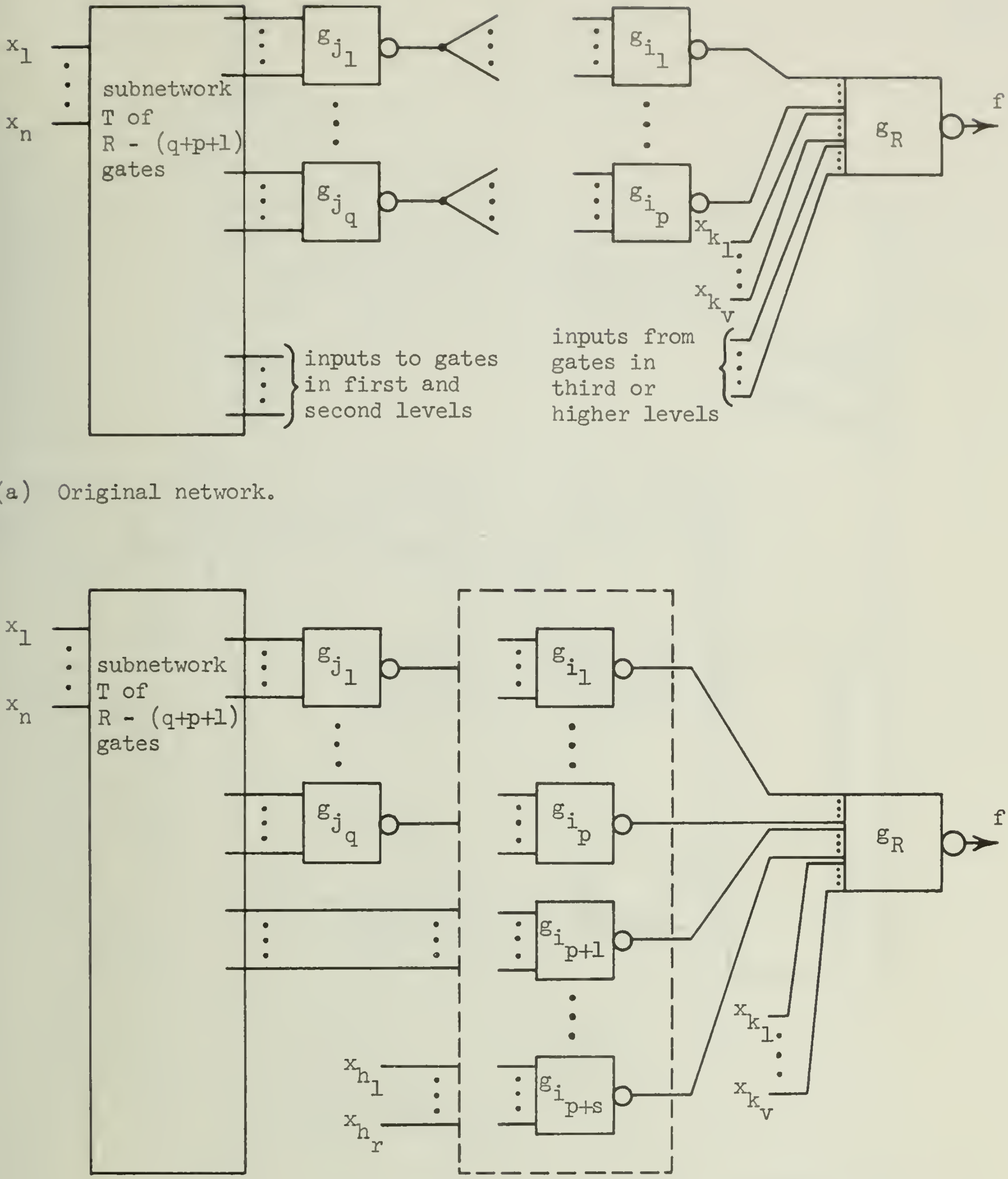
Theorem 3.5 In a single-output, G-minimal negative gate (MOS cell) network, the number of gates (cells) in the second and third levels of the network is less than or equal to $\lceil \log_2(r + 1)\rceil$ where r is the number of external variables in the set $IP(IP(g_R))$ and g_R is the output gate (cell) of the network.

Proof The proof will be by demonstration of the fact that every network of R negative gates can be transformed into a network of $R - (p + q) + \lceil \log_2 (r + 1) \rceil$ negative gates where p, q, and r are, respectively, the number of gates in the second level, the number of gates in the third level, and the number of external variables in the set $IP(IP(g_R))$ of the original network. Thus, if the condition stated in the theorem is not met by a particular network, a network of fewer gates realizing the same function can be derived - implying that the original network is not G-minimal.

In a given single-output network of R negative gates, let g_R denote the output gate, let $x_{k_1},\ldots,x_{k_v}$ denote the external variable inputs to g_R, let $g_{i_1},\ldots,g_{i_p}$ denote the gates in the second level, and let $g_{j_1},\ldots,g_{j_q}$ denote the gates in the third level. Also, let the remainder of the network, consisting of $R - (p + q + 1)$ gates, be designated 'subnetwork T.' (See Fig. 3.3(a).)

It is possible that g_R of the given network has inputs other than external variables and second level gates. Let g be a gate which is such an input to g_R. For each such g: make a duplicate negative gate g_0 having the same inputs and realizing the same function as g, disconnect g from g_R, and connect g_0 to g_R. Each such g_0 becomes a second level gate of the network. Let the new second level gates be denoted $g_{i_{p+1}},\ldots,g_{i_{p+s}}$, and let the set of external variables which are inputs of the second level gates, $g_{i_1},\ldots,g_{i_{p+s}}$, be denoted $x_{h_1},\ldots,x_{h_r}$. (See the network in Fig. 3.3(b).) Obviously, the network still realizes the function f. And $IP(IP(g_R))$ in the original network (Fig.3.3 (a)) is now equivalent to $IP(g_{i_1}) \cup \ldots \cup IP(g_{i_{p+s}})$. Hence the external variables in the set $IP(IP(g_R))$ are $x_{h_1},\ldots,x_{h_r}$.

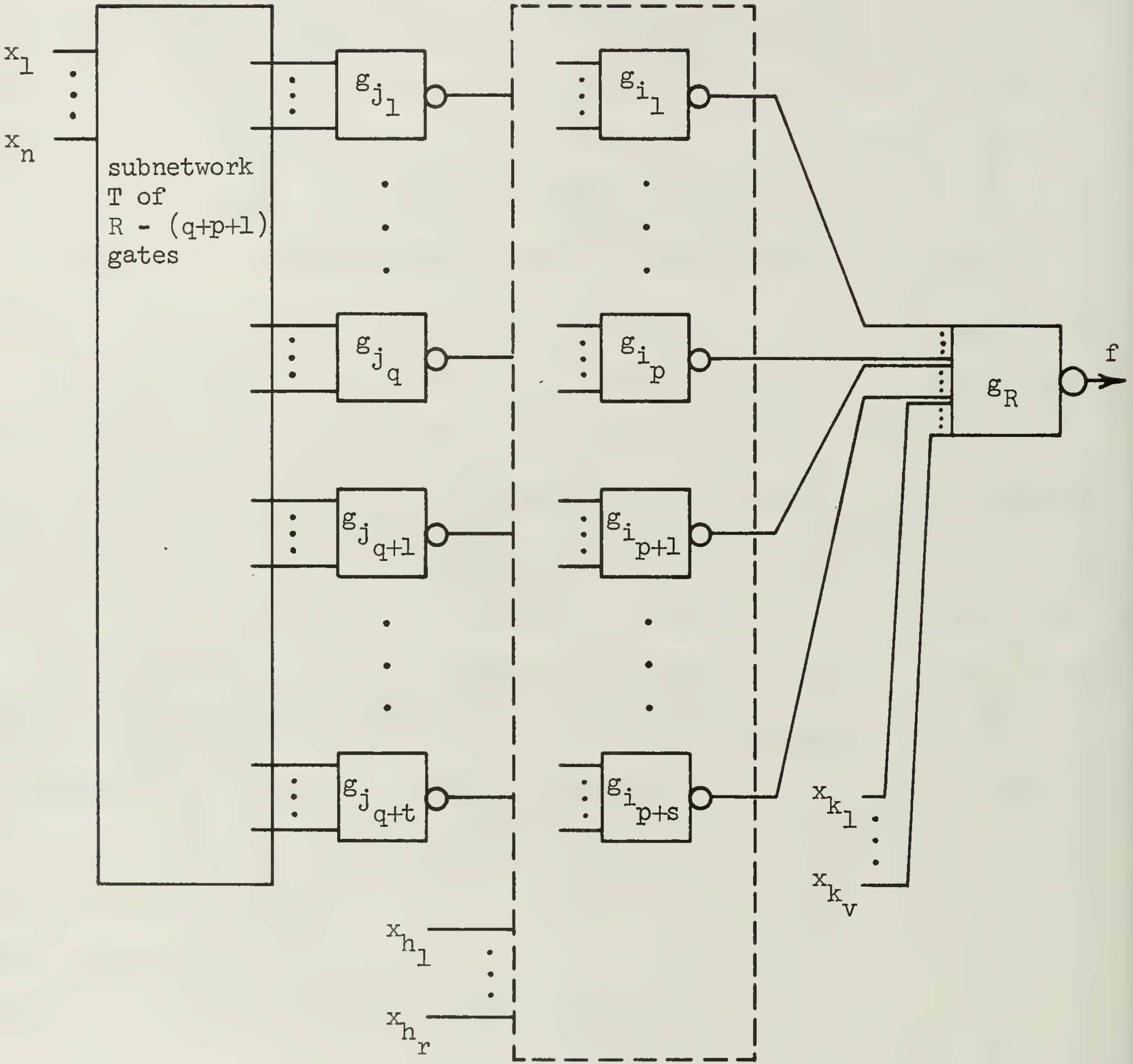
It is now possible that the second level gates, $g_{i_1},\ldots,g_{i_{p+s}}$, have inputs


x_1
x_n
subnetwork T of R - (q+p+1) gates
g_{j_1}
g_{j_q}
g_{i_1}
g_{i_p}
g_R
f
x_{k_1}
x_{k_v}
inputs to gates in first and second levels
inputs from gates in third or higher levels
$g_{i_{p+1}}$
$g_{i_{p+s}}$
x_{h_1}
x_{h_r}

(a) Original network.

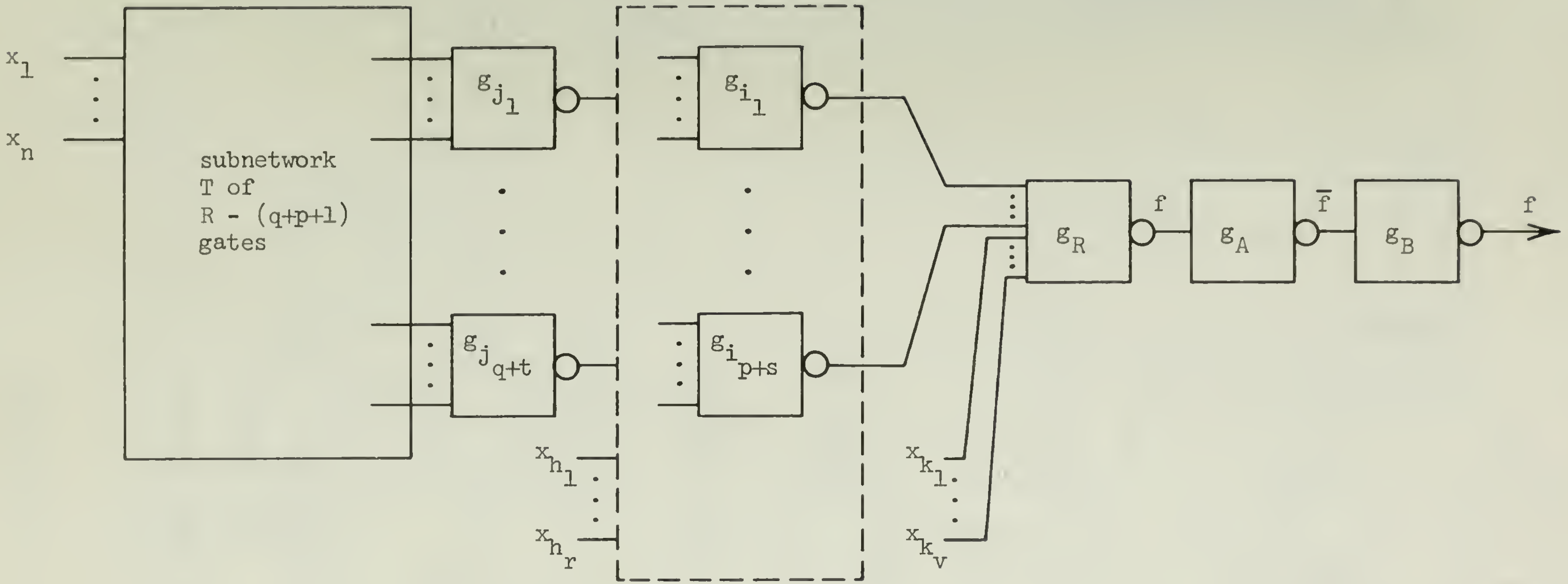
(b) Network after adding new gates to second level.

Fig. 3.3 Illustration of the proof for Theorem 3.5.



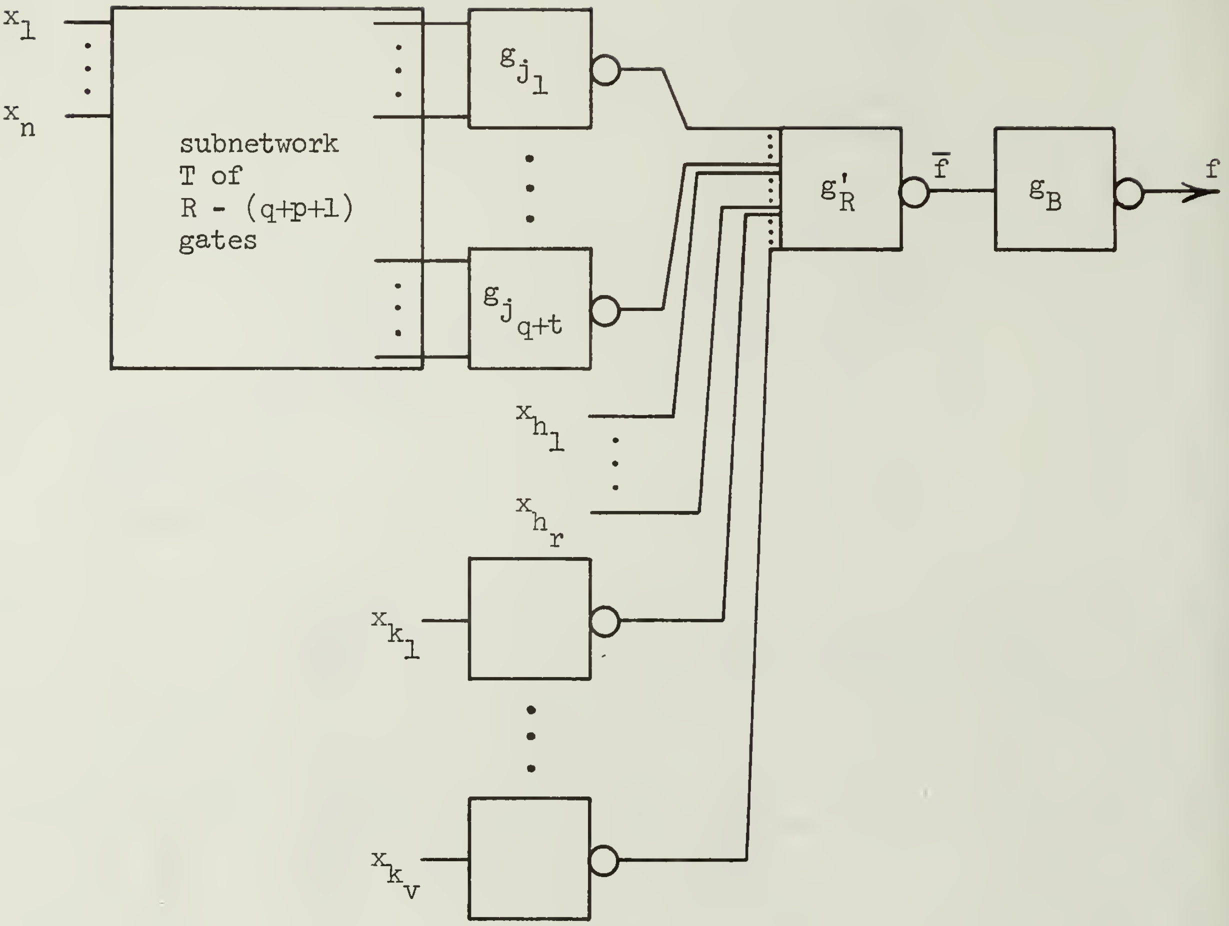
(c) Network after adding new gates to third level.

Fig. 3.3 (Continued)

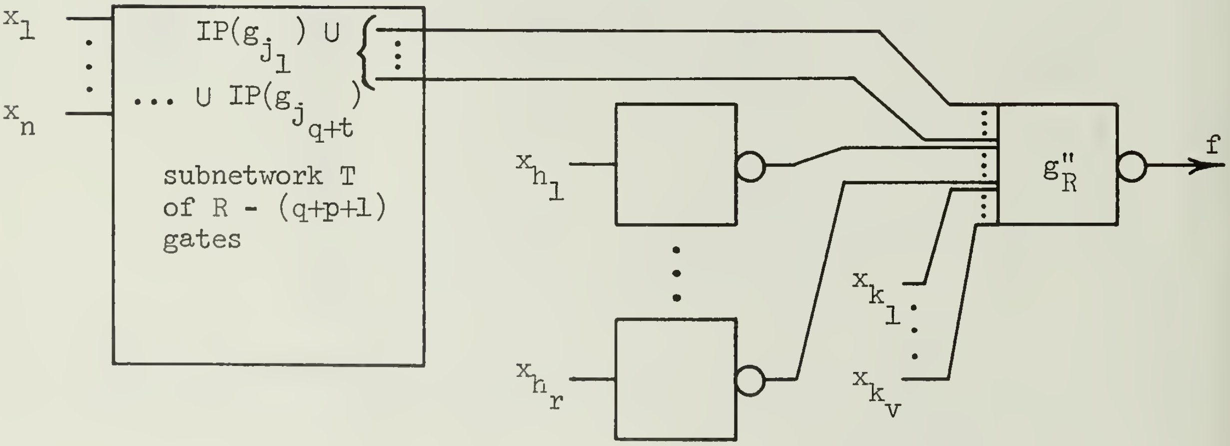


(d) Network after adding two inverters to output.

Fig. 3.3 (Continued)

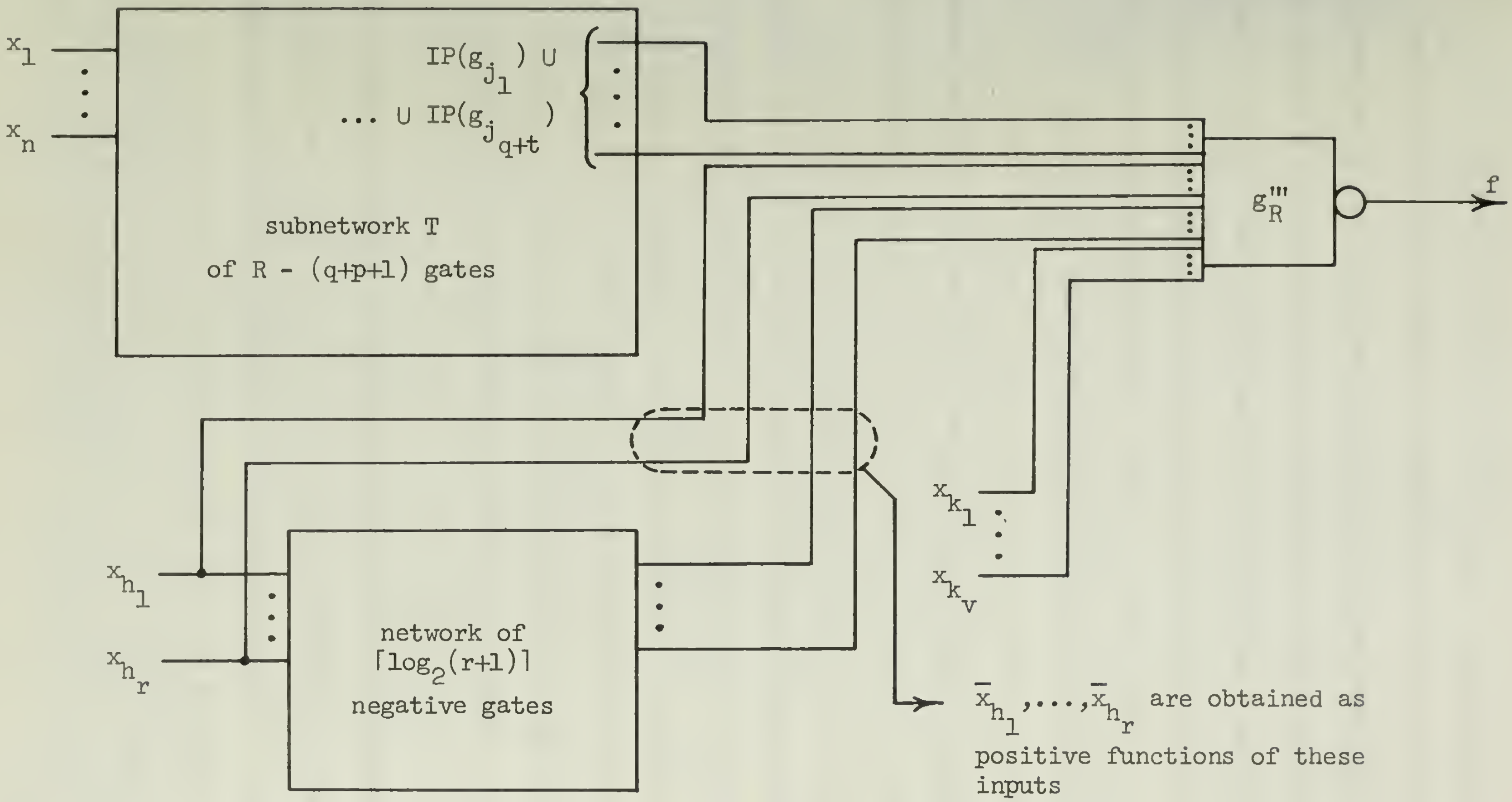


(e) Network after first simplification.



(f) Network after second simplification.

Fig. 3.3 (Continued)



$\bar{x}_{h_1}, \dots, \bar{x}_{h_r}$ are obtained as positive functions of these inputs

(g) Network after final simplification.

Fig. 3.3 (Continued)

other than external variables $x_{h_1},\ldots,x_{h_r}$ and third level gates $g_{j_1},\ldots,g_{j_q}$. Let g be a gate which is such an input to a gate in the second level. For each such g: make a duplicate negative gate g_0 having the same inputs and realizing the same function as g, disconnect g from the second level gates of which it is an input, and connect g_0 to the same second level gates. Each such g_0 becomes a third level gate. Let these new third level gates be denoted $g_{j_{q+1}},\ldots,g_{j_{q+t}}$ (refer to Fig. 3.3(c)). It is obvious that the network still realizes function f and that subnetwork T still consists of R - (q + p + 1) gates.

Next, add two negative gates, g_A and g_B, each serving as a simple inverter, to the output of cell g_R (see Fig. 3.3(d)). Since the function f realized by g_R is obviously a negative function of $x_{k_1},\ldots,x_{k_v}$ and the outputs of g_{i_1}, $\ldots,g_{i_{p+s}}$, f is therefore also a positive function of $\overline{x}_{k_1},\ldots,\overline{x}_{k_v}$, and $\{IP(g_{i_1}) \cup \ldots \cup IP(g_{i_{p+s}})\} = \{x_{h_1},\ldots,x_{h_r}, g_{j_1},\ldots,g_{j_{q+t}}\}$. Furthermore, $\overline{f}$ (realized by g_A) is a negative function of the same variables. Providing one negative gate to obtain the complement of each of $x_{k_1},\ldots,x_{k_v}$, $\overline{f}$ can thus be realized by a new negative gate, g_R', with inputs $\overline{x}_{k_1},\ldots,\overline{x}_{k_v}$, $g_{j_1},\ldots,g_{j_{q+t}}$, $x_{h_1},\ldots,$ x_{h_r} (see Fig. 3.3(e)). Function f is still realized by g_B which inverts the output of g_R'.

Next, since function $\overline{f}$ realized by g_R' is a negative function of its inputs, $\overline{f}$ is also a positive function of $\overline{x}_{h_1},\ldots,\overline{x}_{h_r}$, $x_{k_1},\ldots,x_{k_v}$, and $\{IP(g_{j_1}) \cup \ldots \cup IP(g_{j_{q+t}})\}$. Thus f (realized by g_B) is a negative function of the same variables and can be realized by a new negative gate, g_R'', with inputs $\overline{x}_{h_1}$, $\ldots,\overline{x}_{h_r}$, $x_{k_1},\ldots,x_{k_v}$, and $\{IP(g_{j_1}) \cup \ldots \cup IP(g_{j_{q+t}})\}$, where the functions $\overline{x}_{h_1},\ldots,\overline{x}_{h_r}$ are realized by r new gates with the respective inputs $x_{h_1},\ldots,$ x_{h_r}. (See Fig. 3.3(f).)

As discussed in the proof of Theorem 3.2, r gates are not really necessary to realize the complements of r variables. Again using the results of Markov and Muller, $\bar{x}_{h_1},\ldots,\bar{x}_{h_r}$ can be realized as positive functions of $x_{h_1},\ldots,x_{h_r}$ and the outputs of a loop-free subnetwork of $\lceil \log_2(r+1)\rceil$ negative gates having $x_{h_1},\ldots,x_{h_r}$ as inputs. Thus f can be realized by a negative gate, g_R''', having as inputs these $\lceil \log_2(r+1)\rceil$ negative gates, gates in $IP(g_{j_1}) \cup \ldots \cup IP(g_{j_{q+t}})$, and external variables in $\{x_{h_1},\ldots,x_{h_r}\} \cup \{x_{k_1},\ldots,x_{k_v}\}$ (see Fig. 3.3(g)). This final version of the network consists of $R - (p+q) + \lceil \log_2 (r+1)\rceil$ negative gates: the $R - (q+p+1)$ gates of subnetwork T; the other subnetwork of $\lceil \log_2(r+1)\rceil$ gates; and the output gate, g_R'''.

Q.E.D.

For single-output, G-minimal negative gate (MOS cell) networks with one through ten external variables in the set $IP(IP(g_R))$, Table 3.2 gives the corresponding maximum numbers of second plus third level gates as determined according to Theorem 3.5.

If a negative gate network restricted to two levels is to be synthesized for a given function f, certain external variables can be determined to be necessary inputs to (gates in) the two different levels regardless of the specific implementation selected. The following two theorems, the first of which appears in [Iba 71], give criteria for such a determination. They apply to both completely and incompletely specified functions and to both single-output and multiple-output networks.

Theorem 3.6 Let f be one of the output functions realized by a given 2-level network with external variables $x_1,\ldots,x_n$. If there exists a pair of input vectors, $A,B \in V_n$, identical except that $a_i = 1$ and $b_i = 0$ for exactly one i, $1 \leq i \leq n$, such that $f(A) = 0$ and $f(B) = 1$, then x_i must be an input to the output gate realizing f.

The proof is similar to that of the next theorem.

Theorem 3.7 Let f be one of the output functions realized by a given 2-level network with external variables $x_1,\ldots,x_n$. If there exists a pair of input vectors, $A,B \in V_n$, identical except that $a_i = 1$ and $b_i = 0$ for exactly one i, $1 \leq i \leq n$, such that $f(A) = 1$ and $f(B) = 0$, then x_i must be an input to at least one gate of the second level.

Proof Consider output gate g realizing function f. By Theorem 2.2, at least one input of g must have the values 0 and 1 for the network input vectors A and B, respectively. Since this input can not be an external variable by the property of these input vectors A and B, it must be an input from a gate, g', of the second level. Again by Theorem 2.2, at least one input of g' must have the values 1 and 0 for the input vectors A and B, respectively. The only possible such input is x_i since g' can only have external variables as inputs and since $a_j = b_j$ for every $j \neq i$, $1 \leq j \leq n$.

Q.E.D.

The following four very important theorems and corollaries, taken from published results, correlate switching functions with their minimal negative gate realizations. The validity of these theorems will become evident in the discussions of the corresponding network synthesis algorithms in Section 4. The first two theorems deal only with completely specified functions since the corresponding results for incompletely specified functions are not so simply stated and are more easily given after certain discussions in Section 4.

The first theorem, dealing with networks restricted to two levels, is not stated explicitly in [IM 71], but it can easily be seen to be an immediate consequence of the network synthesis algorithm given there:

Theorem 3.8 The minimum number of negative gates required to realize a completely specified function f of n variables is $D^f(\vec{1},\vec{0}) + 1$ when the network realizing the function is limited to two levels.

The next theorem, appearing in [NTK 72] deals with the number of negative gates in a G-minimal network realizing a given function.

Theorem 3.9 The minimum number of negative gates required to realize a completely specified function f of n variables is:

$$\lceil \log_2(D^f(\vec{1},\vec{0}) + 1)\rceil + 1 .$$

Noting that the inversion degree, $D^f(\vec{0},\vec{1})$, of any function f of n variables can be at most $\lfloor (n + 1)/2 \rfloor$(since the longest directed path through an n-cube includes n + 1 vertices), the following two corollaries are apparent from the two preceding theorems.

Corollary 3.2 Any function of n variables can be realized with at most $\lfloor (n + 1)/2 \rfloor + 1$ negative gates when the network realizing the function is limited to two levels.

Corollary 3.3 Any function of n variables can be realized with at most

$$\lceil \log_2(\lfloor (n + 1)/2 \rfloor + 1)\rceil + 1$$

negative gates.

These two corollaries also apply to incompletely specified functions since every completion of a function f of n variables is, of course, also a function of n variables which, as such, is subject to Theorems 3.8 and 3.9.

The relations in Theorem 3.9 and Corollary 3.3 are consistent with the results (mentioned previously, in part) achieved independently by A.A. Markov [Mar 58] and D.E. Muller [Mul 58] who solved the related problem of finding the minimum number of inverters necessary in a network realizing a given function or set of functions. (A discussion of the relation of this problem to that of synthesizing a multiple-level, G-minimal negative gate network is given in [NTK 72].)

The next several theorems relate a function to be realized by an MOS cell with certain characteristics of the driver of the cell. This information is useful in situations in which cells must be designed under restrictions on the numbers of FET's connected in series and/or parallel within a driver. With these results, lower bounds on numbers of FET's in series or parallel can be determined, for a given function f and associated inputs, without the necessity of actually synthesizing the driver. If the maximum number of FET's in series and in parallel are considered to be, respectively, the maximum number of FET's in any conducting path through the driver and the maximum number of FET's constituting a cut-set of the driver, then these results also apply to drivers containing bridge connections.

The first four theorems deal with functions completely specified with respect to the inputs of the cells which realize them.

Theorem 3.10 If w^f_{FMAX} is the maximum among the weights of minimum false vectors for a completely specified negative function f of n variables,

$x_1,\ldots,x_n$, realized by an MOS cell with inputs $x_1,\ldots,x_n$, then the driver of the cell must have at least w^f_{FMAX} FET's connected in series.

Proof Let $A = (a_1,\ldots, a_n)$ be a minimum false vector of weight k for f, where $a_{i_1},\ldots,a_{i_k} = 1$ and $a_{i_{k+1}},\ldots,a_{i_n} = 0$, such that no other minimum false vector exists with weight greater than k. Thus $k = w^f_{FMAX}$.

Since A is a false vector, there must exist at least one conducting path, p, through the driver of the MOS cell for this input vector. This path can consist only of FET's with a subset of $x_{i_1},\ldots,x_{i_k}$ as inputs; otherwise it would not be conducting for A.

Now consider an input vector B, identical to A except for one component b_{i_ℓ}, $1 \leq \ell \leq k$, which is 0 for B. Since $A > B$ and f is completely specified, B must be a true vector for f in order to be consistent with the assumption that A is a minimum false vector for f. For true input vector B there can exist no conducting path through the driver of the MOS cell. If conducting path p for A consisted only of FET's with a proper subset of $x_{i_1},\ldots,x_{i_{\ell-1}}$, $x_{i_{\ell+1}},\ldots,x_{i_k}$ as inputs, then p would also be a conducting path for B, since every x_{i_j}, $1 \leq j \leq k$, $j \neq \ell$, is 1 for both A and B. Thus, at least one FET with x_{i_ℓ} as its input must exist on path p. Similarly, this is true for every x_{i_ℓ}, $\ell = 1,\ldots,k$. Hence, there must be a series of at least k ($= w^f_{FMAX}$) FET's in the driver of the MOS cell realizing f.

Q.E.D.

Theorem 3.11 For a completely specified negative function f of n variables, $x_1,\ldots,x_n$, there exists an MOS cell with inputs $x_1,\ldots,x_n$ realizing f whose driver has at most w^f_{FMAX} FET's connected in series, where w^f_{FMAX} is the maximum among the weights of minimum false vectors of f.

Proof It is simple to design an MOS cell to realize f with no more than w^f_{FMAX} FET's being connected in series in its driver. The transmission function, $\overline{f}$, realized by the driver of the cell can be expressed as the disjunction of products - one for each false vector of f - of noncomplemented literals corresponding to the 1's in the respective false vectors. This can be reduced to a disjunction of products corresponding to only the minimum false vectors of f as all other products are implied by these.

A driver realizing $\overline{f}$ can be constructed directly from this expression. Corresponding to each product of t literals in the expression, a series connection of t FET's is made - one FET for each respective literal in the product. Making a parallel connection of these serially connected "strings" of FET's results in a structure obviously realizing the transmission function $\overline{f}$.

Since the number of FET's in each serially connected string is, by construction, the number of noncomplemented literals in the expression of the corresponding vector and is therefore identical to the weight of the vector, the maximum number of FET's in series in the resultant MOS cell is the same as the maximum weight, w^f_{FMAX}, among all false vectors.

Q.E.D.

Theorem 3.12 If w^f_{TMIN} is the minimum among the weights of maximum true vectors for a completely specified negative function f of n variables, x_1, $\ldots,x_n$, realized by an MOS cell with inputs $x_1,\ldots,x_n$, then the driver of the cell must have at least n - w^f_{TMIN} FET's connected in parallel (i.e., must have a cut set of at least n - w^f_{TMIN} FET's).

Theorem 3.13 For a completely specified negative function f of n variables, $x_1,\ldots,x_n$, there exists an MOS cell with inputs $x_1,\ldots,x_n$ realizing f whose driver has at most n - w^f_{TMIN} FET's connected in parallel, where w^f_{TMIN} is the minimum among the weights of maximum true vectors of f.

Theorems 3.12 and 3.13 can be proved in a manner similar to the proofs of Theorems 3.10 and 3.11, respectively.

Although these four theorems assure the existence of an MOS cell realization for a given negative function f with the minimum w^f_{FMAX} (driver) FET's in series and one with the minimum n - w^f_{TMIN} FET's in parallel, there may not exist a single realization which possesses both of these characteristics.

Example 3.1 Consider the function f defined by the labelled 4-cube in Fig. 3.4. Since there are no inverse edges in the 4-cube, f is a negative function of $x_1,\ldots,x_4$. For f the minimum false vectors are:

(1100), (1001), and (0101).

The maximum weight, w^f_{FMAX}, among these three vectors is two. Following the construction suggested in the proof of Theorem 3.11, f can be expressed in the form:

$$f = \overline{x_1x_2 \vee x_1x_4 \vee x_2x_4}$$

which corresponds to the MOS cell implementation shown in Fig. 3.5(a). This is one of the possible realizations with only w^f_{FMAX} = 2 FET's in series. Now, from the maximum true vectors for f,

(1010), (0110), and (0011),

w^f_{TMIN} is found to be two. Thus, by Theorem 3.13 and MOS cell realizing f can be constructed which has only two FET's in parallel. Based on the three maximum true vectors, the following expression for negative function f can be obtained:

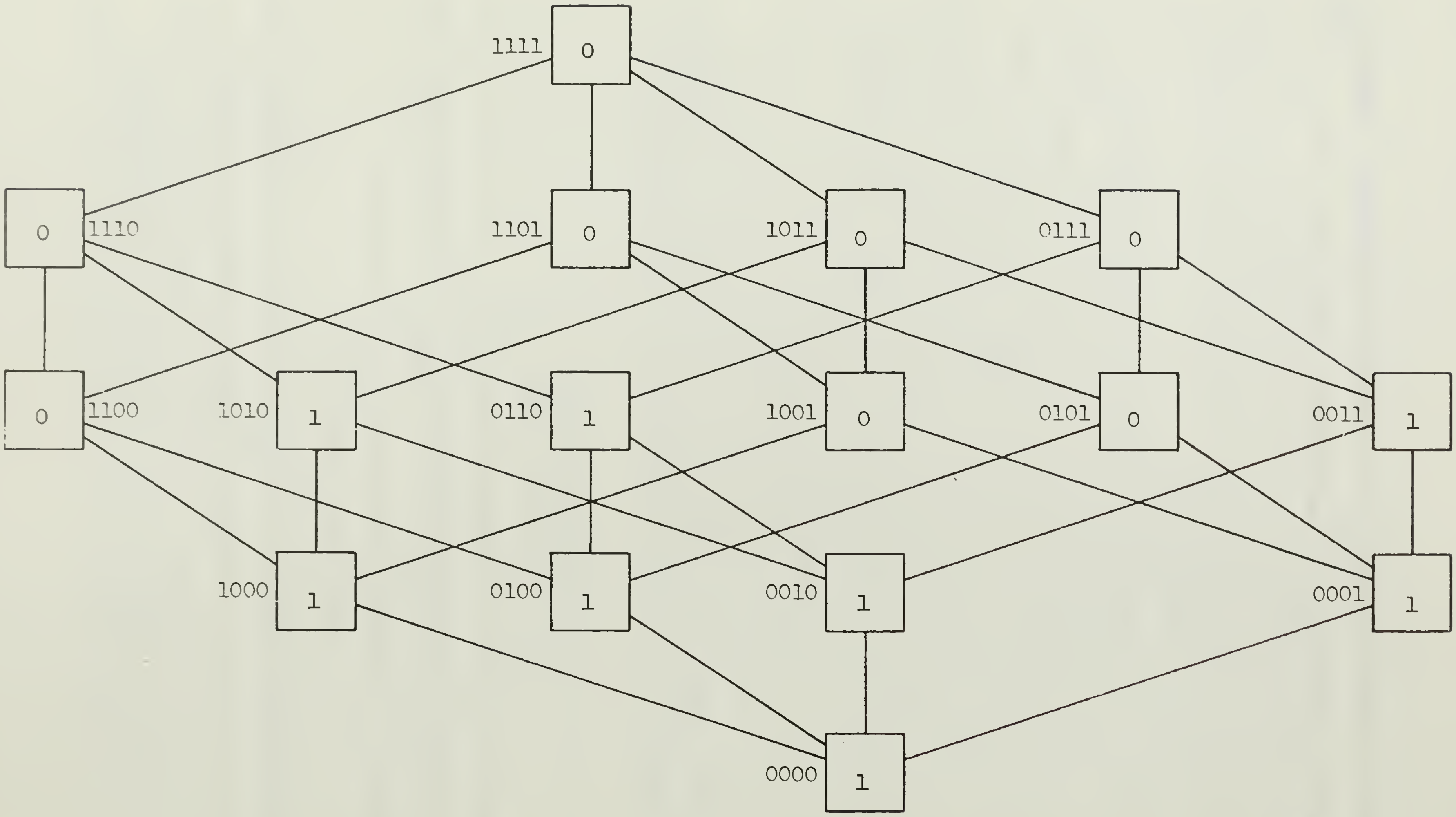
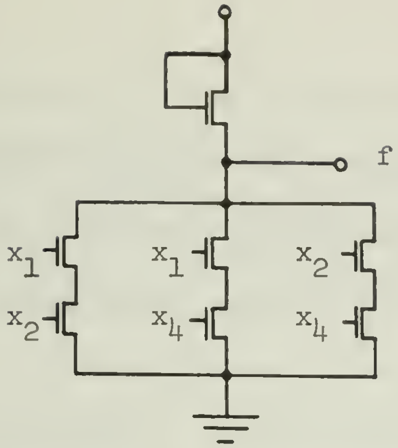
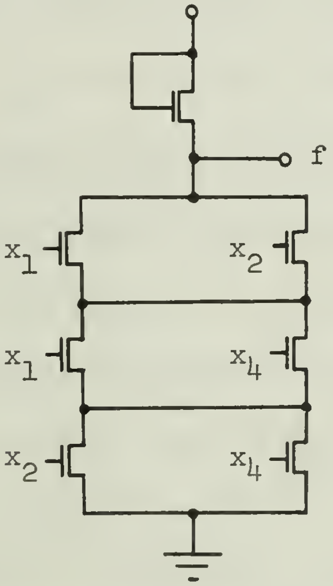


Fig. 3.4 Function f of Example 3.1.



(a) One possible configuration of an MOS cell realizing function f of Fig. 3.4. The driver of this cell has no more than two FET's connected in series.



(b) A second possible configuration of an MOS cell realizing function f of Fig. 3.4. The driver of this cell has no more than two FET's connected in parallel.

Fig. 3.5 MOS cells realizing function f of Fig. 3.4.

$$f = \bar{x}_2\bar{x}_4 \vee \bar{x}_1\bar{x}_4 \vee \bar{x}_1\bar{x}_2 = \overline{(x_2 \vee x_4)(x_1 \vee x_4)(x_1 \vee x_2)}.$$

The corresponding MOS cell with only two FET's in parallel is shown in Fig. 3.5.(b). Note that while the cell in Fig. 3.5(a) has only two FET's in series, it has three in parallel, and while the cell in Fig. 3.5.(b) has only two FET's in parallel, it has three in series. Other possible configurations, including some with bridge connections exist for a cell realizing f, but none has simultaneously at most two FET's in series and at most two in parallel.

The preceding results apply to functions realized by MOS cells which are completely specified negative functions with respect to the inputs of the cell. In general, desired functions will often be incompletely specified with respect to the cell's inputs. Generalizations of the preceding theorems can be made which are applicable in these cases.

Theorem 3.14 For an incompletely specified negative function f of n variables, $x_1,\ldots,x_n$, the driver of every MOS cell with inputs $x_1,\ldots,x_n$ which realizes f has at least W^f_{FUMAX} $(n - W^f_{TUMIN})$ FET's connected in series (parallel) where W^f_{FUMAX} (W^f_{TUMIN}) is the maximum (minimum) among the weights of vectors A satisfying both of the following conditions:

(i) A is a minimum (maximum) vector of the set, Q, consisting of all false (true) vectors of f and every unspecified vector which is less (greater) than at least one false (true) vector of f and not less (greater) than any true (false) vector of f.

(ii) There exists at least one false (true) vector B such that: $A \leq B$ $(A \geq B)$ and $B \not> C$ $(B \not< C)$ for every minimum (maximum) vector C of Q satisfying $w(C) < w(A)$ $(w(C) > w(A))$.

Proof First consider the theorem statement concerning the number of FET's in series. It is sufficient to show that $w^{f'}_{FMAX} \geq w^{f}_{FUMAX}$ for every negative completion f' of f since, by Theorem 3.10, the driver of every MOS cell realizing f' with inputs $x_1,\ldots,x_n$ must have at least $w^{f'}_{FMAX}$ FET's connected in series.

Let A be one of the vectors of weight w^{f}_{FUMAX} which satisfies conditions (i) and (ii) of the theorem. Let B be a false vector of f such that $A \leq B$ and no vector D exists such that: D is a minimum vector of set Q, $w(D) < w(A)$, and $D < B$. At least one such B must exist since A is assumed to satisfy condition (ii). Now consider the following two cases:

Case 1 B is a minimum false vector of f'.

Since $B \geq A$, then $w(B) \geq w(A)$. Hence:

$$w^{f'}_{FMAX} \geq w(B) \geq w(A) = w^{f}_{FUMAX}.$$

Case 2 B is not a minimum false vector of f'.

Then there must exist at least one minimum false vector, C, of f' such that $C < B$. If $w(C) \geq w(A)$ then:

$$w^{f'}_{FMAX} \geq w(C) \geq w(A) = w^{f}_{FUMAX},$$

and the theorem statement is valid. Now, suppose $w(C) < w(A)$. Since $C < B$ and, as a false vector of negative completion f' of f, C can not be less than any true vector of f, then $C \in Q$. Therefore, there must exist a minimum vector, E, of set Q such that $E \leq C$. This implies $w(E) \leq w(C) < w(A)$ and $E \leq C < B$ which contradicts the assumption that A satisfies condition (ii) of the theorem. Hence, no minimum false vector, C, of f' can exist such that $C < B$ and $w(C) < w(A)$.

Since the theorem has been shown to hold in each of these two cases which exhaust all possibilities, the theorem statement concerning the number of FET's in series must be valid.

A similar proof, based on Theorem 3.12, can be made for the part of the theorem statement concerning the number of FET's in parallel.

Q.E.D.

In determining a configuration of an MOS cell realizing an incompletely specified negative function of its inputs, there is freedom both in selecting the negative completion of the function and in choosing the specific configuration realizing the negative completion. Only the latter freedom occurs in the case of completely specified negative functions.

The following theorem proves the existence of negative completions of a function which permit the achievement of the lower bounds on the number of driver FET's in series or parallel given by Theorem 3.14.

Theorem 3.15 For an incompletely specified negative function f of n variables, $x_1,\ldots,x_n$, there exists an MOS cell with inputs $x_1,\ldots,x_n$ which realizes f whose driver has at most w^f_{FUMAX} $(n - w^f_{TUMIN})$ FET's connected in series (parallel) where w^f_{FUMAX} (w^f_{TUMIN}) is the maximum (minimum) among the weights of vectors A satisfying both of the following conditions:

(i) A is a minimum (maximum) vector of the set, Q, consisting of all false (true) vectors of f and every unspecified vector which is less (greater) than at least one false (true) vector of f and not less (greater) than any true (false) vector of f.

(ii) There exists at least one false (true) vector B such that: $A \leq B$

($A \geq B$) and $B \not> C$ ($B \not< C$) for every minimum (maximum) vector C of Q satisfying $w(C) < w(A)$ ($w(C) > w(A)$).

Proof Consider the part of the theorem statement concerning driver FET's in series. The theorem statement concerning FET's in parallel can be demonstrated in a similar manner.

Let T be the set of all vectors A satisfying both conditions (i) and (ii) of the theorem. For every vector B such that B is greater than or equal to at least one vector A, $A \in T$, select B as a false vector of completion f' of f. All other unspecified vectors of f are selected as true vectors of f.

First, f' will be shown to be an actual completion of f.

Consider a false vector, C, of f. Since $C \in Q$, there must exist a vector A_1 satisfying condition (i) such that $A_1 \leq C$. If there does not exist another vector A_2, $A_2 \not= A_1$, such that $A_2 \in T$ and $A_2 \leq C$, then, by condition (ii), A_1 must also be an element of T. In either case, there exists a vector $A \in T$ such that $A \leq C$. Thus, by the method of selecting false vectors of f', C is also a false vector of f'.

Now consider a true vector, C, of f. In order to prove C is also a true vector of f', assume it is not. Then vector C must be greater than or equal to some vector $D \in T$ (by the method of selection of false vectors of f'). This is not possible since D can only be a false vector of f (contradicting the assumption that f is a negative function) or an unspecified vector of f (contradicting the assumption that no such vector in Q is less than a true vector of f).

Since every true vector of f has been shown to be a true vector of f' and every false vector of f has been shown to be a false vector of f', completely specified function f' is thus a completion of f. To prove that f' is, in addition, a negative completion of f, assume it is not. Then there exists at least one pair cf vectors, C and D, such that C is a true vector of f', D is a false vector of f', and $C > D$. However, since D is a false vector of f', $D > A$ for some vector $A \in T$. Hence, $C > A$, and C must also be a false vector of f' (contradicting the assumption that it is a true vector of f').

By condition (i), it is obvious that no two vectors in set T are comparable. Since the set of false vectors for f' consists of all vectors in set T plus all vectors greater than at least one vector in T, set T must be the set of minimum false vectors of f'. Since the largest weight of any vector in set T is w^{f}_{FUMAX} by definition, $w^{f}_{FUMAX} = w^{f'}_{FMAX}$. Thus, by Theorem 3.11, there exists an MOS cell with at most w^{f}_{FUMIN} FET's connected in series.

Q.E.D.

4. OPTIMAL NETWORK SYNTHESIS BASED ON A NEW CONCEPT OF CLUSTERS

New algorithms for the synthesis of both 2-level and multiple-level, G-minimal MOS networks will be presented in this section. These new algorithms are of two types. The first results from an observation of the relationship between Liu's synthesis algorithms (for both 2-level and multiple-level MOS networks) based on 'clusters' and 'stratified structures'[Liu 72] and the n-cube labelling method of Nakamura et al. which is a part of the synthesis algorithms proposed in [NTK 72]. This relationship suggests a new type of 'stratified structure' based upon which new algorithms paralleling each of those of [Liu 72] can be easily derived. Synthesis results of generally the same quality as those of the algorithms of [Liu 72] can be expected.

The second type employs a new concept of 'clusters' (of true and false vectors). In the case of 2-level synthesis, the resulting algorithms are referred to as algorithms based on non-stratified structures. The new algorithms for the multiple-level case are classified as synthesis algorithms based on stratified structures and extended floor (or ceiling) functions. These algorithms (for both 2-level and multiple-level cases) involve somewhat more computation than the corresponding algorithms of Liu, but significantly improved synthesis results can often be obtained while maintaining the simple derivation of MOS cell configurations which is characteristic of algorithms in [Liu 72].

After a review of existing MOS and negative gate network synthesis methods, this section's discussion will be divided into two parts. Section 4.1 will be concerned with the synthesis of G-minimal 2-level networks of MOS cells (negative gates), and Section 4.2 will deal with the synthesis of G-minimal multiple-level networks. In these two subsections, certain network synthesis algorithms and related algorithms of [Liu 72] and [NTK 72] will first be covered in some detail in order to provide the necessary background for the presentation of the new algorithms later in the subsections (some of this background will also be vital to the discussion of another topic in Section 7).

Before beginning the presentation of the new synthesis algorithms, let us review existing algorithms which have appeared in the literature. This will provide the reader with some basis for assessing the relative advantages and disadvantages of the new synthesis algorithms.

In [IM 69] which is the first paper on the synthesis of MOS networks, Ibaraki and Muroga, recognizing the promise of negative gate network design as an approach to the design of networks of MOS cells, proposed and solved the problem of synthesizing a 2-level, G-minimal, negative gate network.

The algorithm which they presented in [IM 71] (improved over that given in [IM 69]) is based on a truth table and a simple graph generated from it. Compared to later algorithms, this one might be somewhat more difficult to apply by hand, but there should be no difficulties encountered in carrying out the calculations by computer.

An important feature of the algorithm is that it permits incompletely specified functions to be selected as desired outputs of the G-minimal networks to be synthesized. [IM 71] also gives a simple extension of the algorithm which allows G-minimal, multiple-output networks to be synthesized — although only under the restriction that any output function realized by a second level gate must be completely specified (output functions realized by first level gates may be either completely or incompletely specified). An extension of the algorithm to include these restricted cases might involve the solution of a type of covering problem.

Being mainly concerned with minimizing the number of negative gates in a 2-level network, [IM 71] does not discuss how to obtain from the truth tables the internal structures of the corresponding MOS cells implementing the negative gate networks.

Ibaraki[Iba 71] extended the algorithm in [IM 71] to obtain 2-level, MOS cell networks minimizing any given monotone nondecreasing cost function of the number of gates and number of connections. Thus, the algorithm has great generality, and it can yield, among others, G-minimal, GI-minimal, I-minimal, and IG-minimal 2-level networks.

As in [IM 71], incompletely specified functions may be chosen as output functions of the networks to be synthesized. The algorithm can synthesize optimal 2-level, multiple-output networks under the restriction that all output functions must be realized by gates of the first level.

Not surprisingly for an algorithm of this magnitude of generality, it requires the solution of two minimum covering problems to obtain an optimal network of MOS cells, and certain parts of the algorithm involve calculations which may grow quickly in size as the numbers of external variables or output functions are increased.

The algorithm in [Iba 71] also gives some consideration to reducing the number of FET's in the drivers of the MOS cells of the networks. A method for finding an 'SCMS' (simplest complemented minimum sum: one with the minimum number of literals among those with the minimum number of product terms) or an 'SCMP' (simplest complemented minimum product) expression for a cell's output in terms of its inputs is given which involves a minimum covering problem. These expressions may of course be further factored, in general, to produce expressions of fewer literals (this corresponds to fewer FET's used in the implemented cell).

Later, Nakamura, Tokura, and Kasami[NTK 72] and Liu[Liu 72][Liu 75] independently developed algorithms for synthesizing both 2-level and multiple-level, G-minimal, negative gate networks.

In [NTK 72], the authors first showed the similarity of the problem of synthesizing multiple-level, G-minimal, negative gate networks to the problem of minimizing the number of NOT elements (inverters) in a network — a problem solved independently by Markov[Mar 58] and Muller[Mul 58] (see [Ake 68] or [Mur 71], pp. 416-419, for simplified discussions of their results). Then, an algorithm, based on 'n-cube labelling' (to be discussed more later), is obtained for the synthesis of multiple-level, single-output, G-minimal, negative gate networks. This is extended to the multiple-output case. While the results of the extended algorithm can not be guaranteed to be truly G-minimal, they are G-minimal under the restriction that output function f_i is realized by a specific gate (of the generalized form of a feed-forward network) g_{k_i}, i = 1,...,m, chosen prior to the synthesis operation. For both of these algorithms, provisions are made which permit incompletely specified functions to be selected as network output functions.

Following these algorithms, [NTK 72] develops an algorithm for synthesizing k-level-restricted, single-output, G-minimal, negative gate networks,† though this is actually more complex than the non-level-restricted case. The 2-level synthesis problem can thus be solved as a special case (k = 2) of those falling within the capability of this general algorithm.

†The statement of this algorithm in [NTK 72] inadvertently omits one condition necessary for guaranteeing an optimal result. The necessary condition, however, should soon become evident to a user of the algorithm.

In [Liu 72] Liu presents algorithms for the synthesis of 2-level and multiple-level, single-output, G-minimal networks for completely specified functions. [Liu 75] essentially repeats the results in [Liu 72] concerning the two-level case.† A very significant feature of Liu's synthesis algorithms is the ease with which Boolean expressions are derived which directly represent the internal structures of the MOS cells of the implemented network. Previously mentioned synthesis methods (those in [IM 71], [Iba 71], and [NTK 72]) basically operate in two phases: first, the output functions to be realized by every cell of the network are obtained and represented in truth table forms (either standard truth tables or n-cubes); next, from these representations, negative completions are chosen for each of the functions, and then the actual cell configurations are determined. The algorithms proposed in [Liu 72] and [Liu 75] first create a special structure, called the 'stratified structure', which is really a unique partitioning of the true and false vectors for a given function into 'clusters'. From these 'clusters', negative expressions (and thus, MOS cell configurations can be directly obtained, in the 2-level case for each of the cells in the network. In the multiple-level case, also based on 'clusters' of the 'stratified structure', slightly more work (but still a very small amount relative to comparable steps of the other algorithms) is required to obtain the negative expressions for the cells. Since the creation of the stratified structure is of approximately the same difficulty or simpler than the first phases of each of the preceding synthesis methods, Liu's method can generally obtain a final result with significantly less effort.

[Liu 72] also presents a few ideas for reducing the numbers of FET's in

†[Liu 75] mentions a simple extension to the case of incompletely specified functions.

networks synthesized by Liu's multiple-level algorithm and a method for making the complexities of cells in a G-minimal network more nearly uniform.

[Lai 76] first gives an algorithm to obtain G-minimal, single-output, multiple-level, irredundant MOS networks for completely specified functions. Irredundant MOS networks are those from which no FET or group of FET's can be removed without causing at least one network output to be in error. Lai's algorithm is later extended to the case of single-output networks for incompletely specified functions. Then, further extensions of the algorithm are discussed which, for given sets of completely specified functions, obtain irredundant, multiple-output networks which are G-minimal under either of the following conditions: (i) the correspondence between output functions and the gates of the generalized form of a feed-forward network is fixed in advance of the synthesis; (ii) all output cells are assumed to be in the first level of the network (i.e., have no intercell connections to other cells). It is mentioned in [Lai 76] that these synthesis methods for multiple-output networks can also be modified to deal with incompletely specified output functions in a manner similar to that given (in [Lai 76]) in the case of single-output networks.

While the synthesis algorithm of [Lai 76] involves a greater amount of calculation than either of the corresponding algorithms of [Liu 72] or [NTK 72] (Lai's algorithm employs calculations similar to those used in the algorithm of [NTK 72] as well as additional calculations), it is invaluable whenever a G-minimal, irredundant network (which is also diagnosable, as will be discussed in Section 7) is desired.

In the beginning of Section 4.1.1 a new concept of clusters will be proposed which can often lead (as will be seen) to improved results (in terms of cells with simplified internal configurations) with a relatively small increase in algorithm complexity over those of [Liu 72].

4.1 Synthesis of G-Minimal, 2-Level Networks

What is referred to as the synthesis of a 2-level network for a function f (or group of functions, $f_1,\ldots,f_m$) is actually the synthesis of a network restricted to at most two levels which realizes function f (functions $f_1,\ldots,$ f_m). Thus, the result of an algorithm for the synthesis of G-minimal, 2-level networks can be a network of 2, 1, or 0 levels (e.g., a G-minimal network of zero levels would occur for a function such as $f = x_1$).

The importance of designing a 2-level network for a given function or set of functions is mainly to minimize the propagation delay between receiving the input signals and producing the output signals. While a 2-level negative gate network theoretically accomplishes this by minimizing the maximum number of gates between any input and output, in practice, the generally large size and complexity of an output cell in an implemented 2-level, MOS cell network may be such that its switching time is considerably longer than that for any of the MOS cells which may be employed in a multiple-level network realizing the same function(s).

4.1.1 Synthesis methods for 2-level networks based on stratified structures

This section presents what will be classified as'synthesis algorithms based on stratified structures.' This includes Liu's algorithm for the synthesis of 2-level networks and similar, but new, algorithms which are closely related to the n-cube labelling proposed by Nakamura et al. In addition to pointing out the relationships between the algorithms of Liu and Nakamura et al., this discussion will establish a basis for the presentation in Section

4.1.2 of improved 2-level synthesis algorithms based on a new concept of clusters and corresponding 'non-stratified structures'.

The following definitions and properties are important for the discussion of synthesis algorithms based on stratified structures.[†] These same properties will also be seen later to be useful in describing synthesis algorithms based on non-stratified structures.

Definition 4.1.1.1 The α-term and β-term of an input vector are, respectively, the product of complemented literals corresponding to the 0's in the input vector, and the product of uncomplemented literals corresponding to the 1's in the input vector. As special cases, the α-term of $\vec{1}$ and the β-term of $\vec{0}$ are defined to be 1. The product, $\alpha \cdot \beta$, is called the minterm of the input vector.

The concept of 'clusters' appears in [Liu 72], but , in preparation for the later presentation of improved synthesis algorithms, it is necessary to generalize this concept as follows:

Definition 4.1.1.2 A true cluster (false cluster), with respect to a function f, is a set of true (false) vectors of function f such that for every pair of vectors, A_i and A_j, of the set which satisfies $A_i > A_j$, every vector A_k satisfying

$$A_i > A_k > A_j$$

is also a member of the set. Both true clusters and false clusters may be referred to, simply, as clusters.

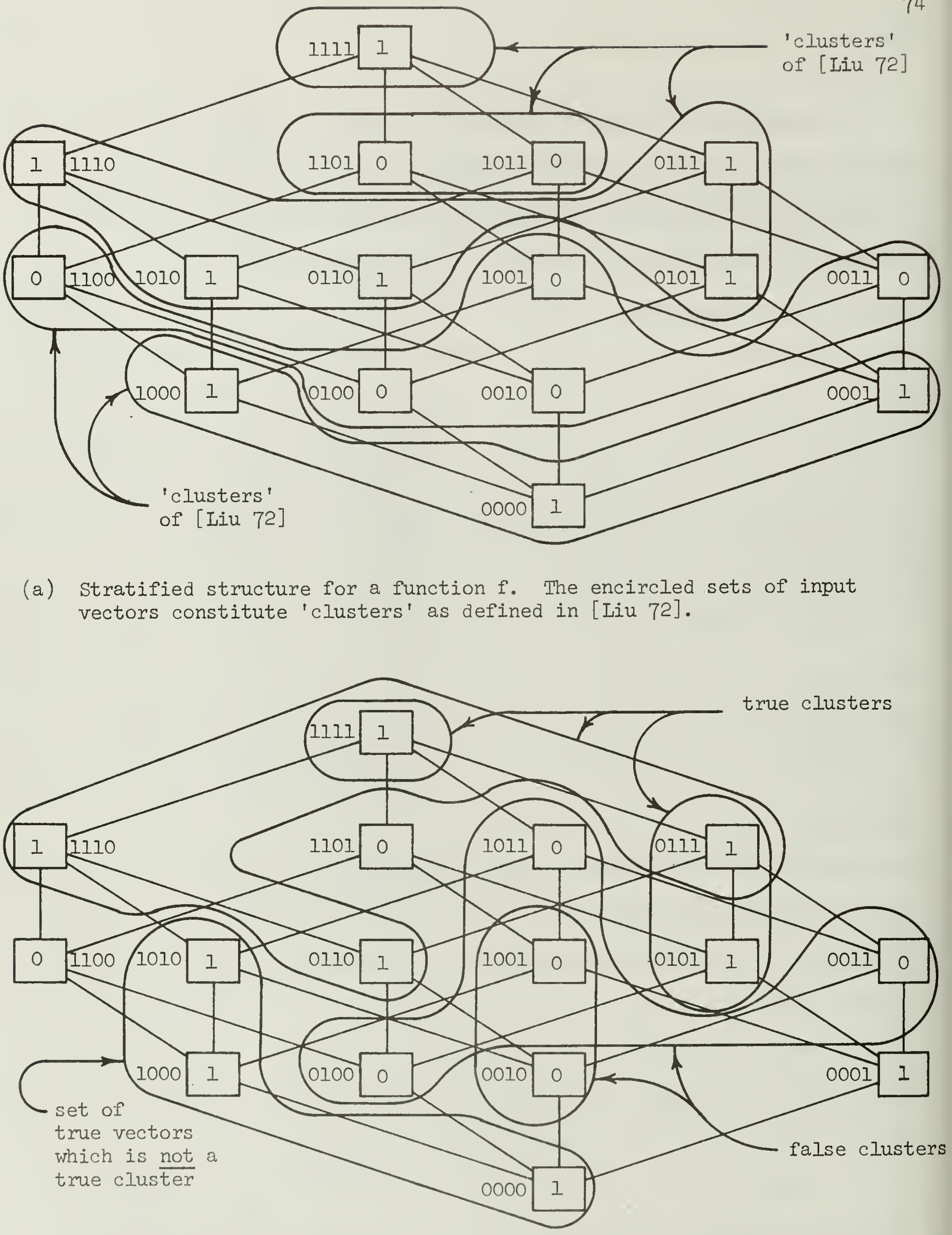
[†] Some of the notation as well as some of the definitions and properties given here are borrowed or modified from that in [Liu 72]. There are some important differences, however, which must be noted to avoid possible confusion.

It is important to note that the term 'cluster' as defined above is more general than the same term defined in [Liu 72] (and [Liu 75]).[†] In [Liu 72], a (true or false) 'cluster' is defined only in the context of a 'stratified structure.' A stratified structure, which will be defined shortly, is actually a unique partitioning of the specified vectors of a given function into several sets of true or false vectors. (For example, the partitioning of input vectors in Fig. 4.1.1.1(a) is an actual stratified structure for the function f represented by the 3-cube). Each of these sets is then defined in [Liu 72] to be a (true or false) 'cluster', Hence, 'clusters' defined in this (Liu's) manner are only meaningful with respect to the stratified structure of the function under discussion.

(True or false) clusters in Definition 4.1.1.2, however, are defined only with respect to a given function and not with respect to any particular grouping or partitioning of input vectors. While it is true (and will be seen) that a 'cluster' of [Liu 72] is a cluster under Definition 4.1.1.2, the converse is not true. Hence, each set of encircled input vectors in the stratified structure illustrated by Fig. 4.1.1.1(a) satisfies the definitions of a cluster in both [Liu 72] and Definition 4.1.1.2, while sets of input vectors such as {(1111)}, {(1111),(1110),(0111),(0110)}, {(0111),(0101)}, {(1001),(0010)}, {(1011),(1001),(0011),(0100)}, etc. (see Fig. 4.1.1.1(b)) are clusters only in the sense of Definition 4.1.1.2. Henceforth, unless otherwise stated, the use of the term 'cluster' will indicate a cluster according to Definition 4.1.1.2 (even during discussions of stratified structures).

An example of a set of true vectors which is not a true cluster (refer to Fig. 4.1.1.1(b)) is:

[†] [Lai 75] contains yet other definitions of clusters.



(a) Stratified structure for a function f. The encircled sets of input vectors constitute 'clusters' as defined in [Liu 72].

(b) Examples of sets of input vectors which are clusters in the sense of Definition 4.1.1.2. (Note that encircled sets in (a) also satisfy Definition 4.1.1.2.)

Fig. 4.1.1.1 Comparison of 'clusters' as defined in [Liu 72] vs. clusters as defined in Definition 4.1.1.2.

$$\{(1010),(1000),(0000)\}.$$

It is not a valid true cluster since false vector (0010) satisfies:

$$(1010) > (0010) > (0000).$$

Note that a subset of the vectors in a cluster need not contain vectors comparable to any of those in the remainder of the cluster, i.e., in a sense, a cluster need not be 'connected'. In the case of the false cluster $\{(1011), (1001),(0011),(0100)\}$ shown in Fig. 4.1.1.1(b), false vector (0100) is incomparable to the other three vectors of the cluster.

Definition 4.1.1.3 The minimum and maximum vectors of a cluster are called, respectively, floor vectors and ceiling vectors of the cluster.

In the false cluster $\{(1011),(1001),(0011),(0100)\}$ of Fig. 4.1.1.1(b), vectors (1011) and (0100) are ceiling vectors of the cluster while (1001), (0011), and (0100) are floor vectors. Note that vector (0100) is both a ceiling and floor function.

Definition 4.1.1.4 The ceiling function of a cluster with ceiling vectors $A_1,\ldots,A_k$ is given by:

$$\alpha_1 \vee \ldots \vee \alpha_k,$$

where α_i is the α-term of ceiling vector A_i. The floor function of a cluster with floor vectors $B_1,\ldots,B_j$ is given by:

$$\beta_1 \vee \ldots \vee \beta_j$$

where β_i is the β-term of floor vector B_i.

For example, the floor function of the false cluster $\{(1011),(1001),(0011), (0100)\}$ in Fig. 4.1.1.1(b) is:

$$x_2 \vee x_1x_4 \vee x_3x_4,$$

corresponding to its floor vectors, (0100), (1001), and (0011). The ceiling function of the same cluster is:

$$\bar{x}_2 \vee \bar{x}_1\bar{x}_3\bar{x}_4,$$

corresponding to its ceiling vectors, (1011) and (0100).

<u>Theorem 4.1.1.1</u> Let $Q = \{A_1,\ldots,A_\ell\}$ be a cluster of vectors with respect to a function f. Let $\alpha_{j_1} \vee \ldots \vee \alpha_{j_p}$ and $\beta_{k_1} \vee \ldots \vee \beta_{k_q}$ be the ceiling and floor functions of the cluster, respectively, and let $A_1 \vee \ldots \vee A_\ell$ denote the disjunction of minterms of the vectors $A_1,\ldots,A_\ell$ in set Q.[†] Then: if Q is a true cluster,

$$A_1 \vee \ldots \vee A_\ell = (\alpha_{j_1} \vee \ldots \vee \alpha_{j_p})(\beta_{k_1} \vee \ldots \vee \beta_{k_q}) \subseteq f;$$

or, if Q is a false cluster,

$$A_1 \vee \ldots \vee A_\ell = (\alpha_{j_1} \vee \ldots \vee \alpha_{j_p})(\beta_{k_1} \vee \ldots \vee \beta_{k_q}) \subseteq \bar{f}.$$

<u>Proof</u> Consider the case when the set Q is a true cluster. Obviously, the disjunction of minterms corresponding to the vectors of the cluster implies f since the vectors are all true vectors of f. Next, let $A_{j_t} \in Q$, $1 \leq t \leq p$, be a ceiling vector of the cluster with α-term α_{j_t}, and let $B_{k_u} \in Q$, $1 \leq u \leq q$ be a floor vector with β-term β_{k_u}. Now, since $\alpha_{j_t} = 1$ for vectors A such that $A \leq A_{j_t}$ and $\beta_{k_u} = 1$ for vectors B such that $B \geq B_{k_u}$, $\alpha_{j_t}\beta_{k_u} = 1$ for a vector C if and only if

$$A_{j_t} \geq C \geq B_{k_u}.$$

Since (1) every vector $C \in Q$ must obviously satisfy this condition for at least one pair of A_{j_t}, $1 \leq t \leq p$, and B_{k_u}, $1 \leq u \leq q$, and (2) every vector C

[†] To eliminate the need for introducing another notation, A_i will be used both to denote a vector and its corresponding minterm. The intended usage should be clear from the context whenever the notation appears.

satisfying this condition must be a member of Q by the definition of a cluster, it can be concluded that:

$$A_1 \vee \dots \vee A_\ell = (\alpha_{j_1} \vee \dots \vee \alpha_{j_p})(\beta_{k_1} \vee \dots \vee \beta_{k_q}).$$

A similar argument can be made when set Q is a false cluster.

Q.E.D.

As an example, consider the false cluster {(1011),(1001),(0011),(0100)} of Fig. 4.1.1.1(b):

$$x_1\bar{x}_2x_3x_4 \vee x_1\bar{x}_2\bar{x}_3x_4 \vee \bar{x}_1\bar{x}_2x_3x_4 \vee \bar{x}_1x_2\bar{x}_3\bar{x}_4$$
$$= (x_2 \vee x_1x_4 \vee x_3x_4)(\bar{x}_2 \vee \bar{x}_1\bar{x}_3\bar{x}_4) \subseteq \bar{f}.$$

The true cluster {(1111),(1110),(0111),(0110)} of the same figure is a special case in which the ceiling function is the constant 1:

$$x_1x_2x_3x_4 \vee x_1x_2x_3\bar{x}_4 \vee \bar{x}_1x_2x_3x_4 \vee \bar{x}_1x_2x_3\bar{x}_4 = (x_2x_3)(1) \subseteq f.$$

It is clear from the preceding theorem that a function f is implied by the disjunction of products of ceiling and floor functions for true clusters with respect to f. Furthermore, if every true vector is a member of at least one of these true clusters, the disjunction is an expression of f itself.

Corollary 4.1.1.1 Let $Q = \{A_1, \dots, A_\ell\}$ be the set of all true vectors (false vectors) or a function f, and let $Q_1, \dots, Q_r$ be sets of vectors which represent true clusters (false clusters) with respect to f such that

$$Q = Q_1 \cup \dots \cup Q_r.$$

Let $A_{i,1}, \dots, A_{i,s_i}$ be the minterms corresponding to vectors in set Q_i, and let $\alpha_{i,1} \vee \dots \vee \alpha_{i,p_i}$ and $\beta_{i,1} \vee \dots \vee \beta_{i,q_i}$ be the ceiling and floor functions, respectively, of the cluster represented by set Q_i. Then:

$$A_{1,1} \vee \cdots \vee A_{1,s_1} \vee \ \cdot \ \cdot \ \cdot \ \vee A_{r,1} \vee \cdots \vee A_{r,s_r} = A_1 \vee \cdots \vee A_\ell$$
$$= (\alpha_{1,1} \vee \cdots \vee \alpha_{1,p_1})(\beta_{1,1} \vee \cdots \vee \beta_{1,q_1}) \vee \cdots \vee$$
$$(\alpha_{r,1} \vee \cdots \vee \alpha_{r,p_r})(\beta_{r,1} \vee \cdots \vee \beta_{r,q_r}) = f \quad (= \overline{\overline{f}}).$$

Suppose true clusters $Q_1,\ldots,Q_r$, having the properties mentioned in Corollary 4.1.1.1, have been chosen for a given function f, and let f be expressed in terms of floor functions and ceiling functions of $Q_1,\ldots,Q_r$ as in Corollary 4.1.1.1. If u_i is defined as the complement of the floor function for true cluster Q_i, $1 \leq i \leq r$, functions $u_1,\ldots,u_r$ and f can be expressed as follows:

$$u_1 = \overline{\beta_{1,1} \vee \cdots \vee \beta_{1,q_1}}$$
$$\vdots$$
$$u_r = \overline{\beta_{r,1} \vee \cdots \vee \beta_{r,q_r}}$$
$$f = (\alpha_{1,1} \vee \cdots \vee \alpha_{1,p_1})\overline{u}_1 \vee \cdots \vee (\alpha_{r,1} \vee \cdots \vee \alpha_{r,p_r})\overline{u}_r$$

Since the u_i are complements of expressions containing only uncomplemented literals and since f is given as an expression containing only complemented literals, then $u_1,\ldots,u_r$, and f are all negative functions of the variables in their respective expressions. Each u_i is a negative function of $x_1,\ldots,x_n$, and f is a negative function with respect to $x_1,\ldots,x_n$, $u_1,\ldots,$ u_r (possibly with some variables being only dummy variables).

This suggests a 2-level negative gate realization of f with negative gates $g_1,\ldots,g_r$ of the second level realizing functions $u_1,\ldots,u_r$, respectively. In the special case when the floor function of the $i^{\underline{th}}$ true cluster is the constant 1, u_i is the constant 0 and does not require a gate g_i.

Consider the function f expressed in the form of a labelled n-cube in Fig. 4.1.1.2. The two encircled true clusters correspond to the following two sets of true vectors:

$$Q_1 = \{(110),(011),(100)\}$$
$$Q_2 = \{(111)\}$$

Following the preceding discussion concerning the construction of a 2-level network to realize function f, functions u_1 and u_2 of the second level gates g_1 and g_2 and network output function f are given by:

$$u_1 = \overline{x_1 \vee x_2x_3}$$
$$u_2 = \overline{x_1x_2x_3}$$
$$f = (\bar{x}_3 \vee \bar{x}_1)\bar{u}_1 \vee (1)\bar{u}_2 = \overline{(x_1x_3 \vee u_1)u_2}.$$

From these expressions, the configurations of the corresponding MOS cells are easily determined. The resultant 2-level network of MOS cells is shown in Fig. 4.1.1.3.

Also using Corollary 4.1.1.1, a 2-level network can easily be constructed for a function f based on false clusters. In this case, letting the functions realized by the second level gates be

$$u_1 = (\alpha_{1,1} \vee \cdots \vee \alpha_{1,p_1}$$
$$\vdots$$
$$u_r = (\alpha_{r,1} \vee \cdots \vee \alpha_{r,p_r}),$$

function $\bar{f}$ can be expressed, according to Corollary 4.1.1.1, as follows:

$$\bar{f} = u_1(\beta_{1,1} \vee \cdots \vee \beta_{1,q_1}) \vee \cdots \vee u_r\,(\beta_{r,1} \vee \cdots \vee \beta_{r,q_r}).$$

Again, $u_1,\ldots,u_r$, and f are seen to be negative functions of the variables

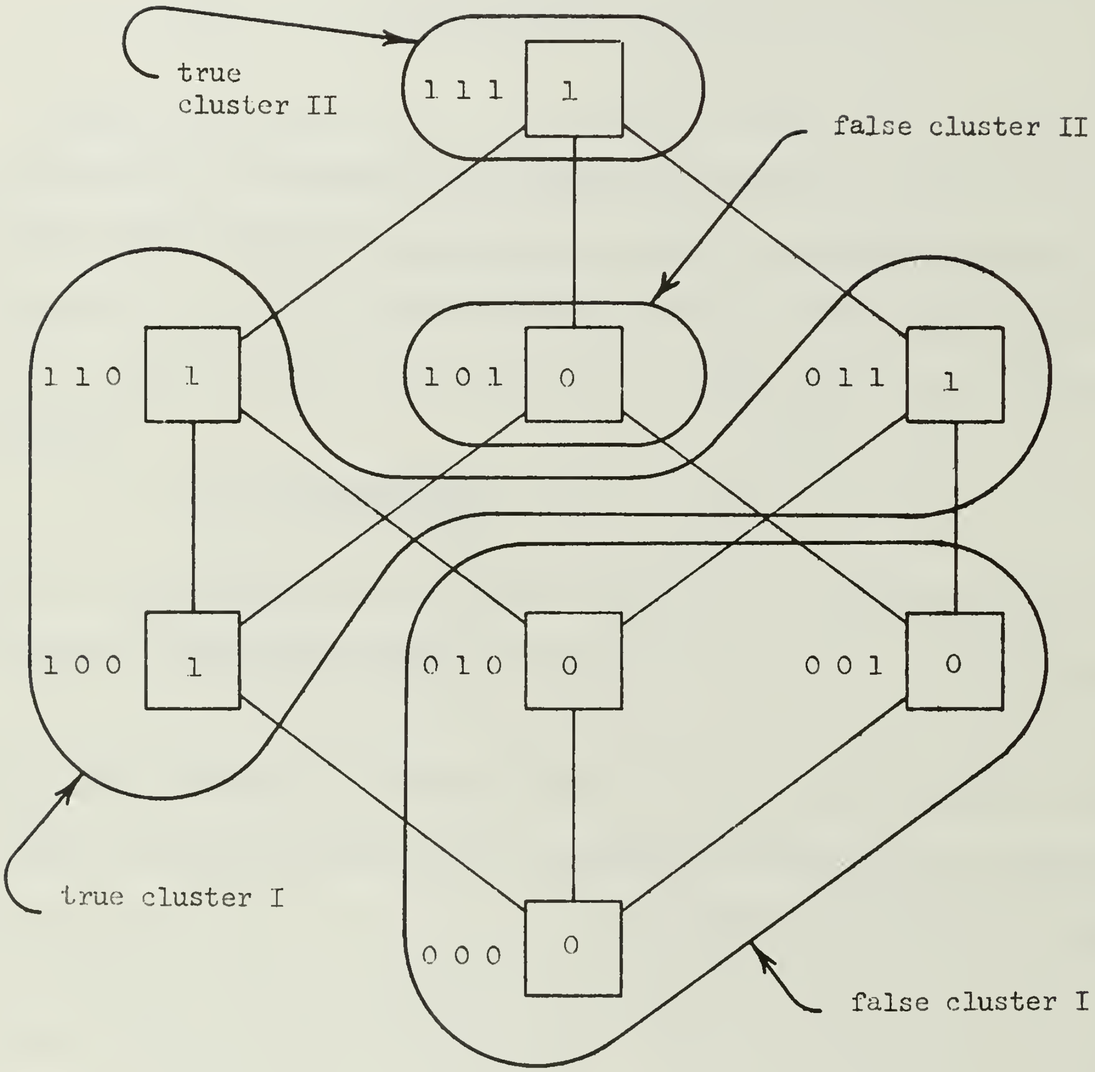


Fig. 4.1.1.2 True and false clusters chosen for a function f.

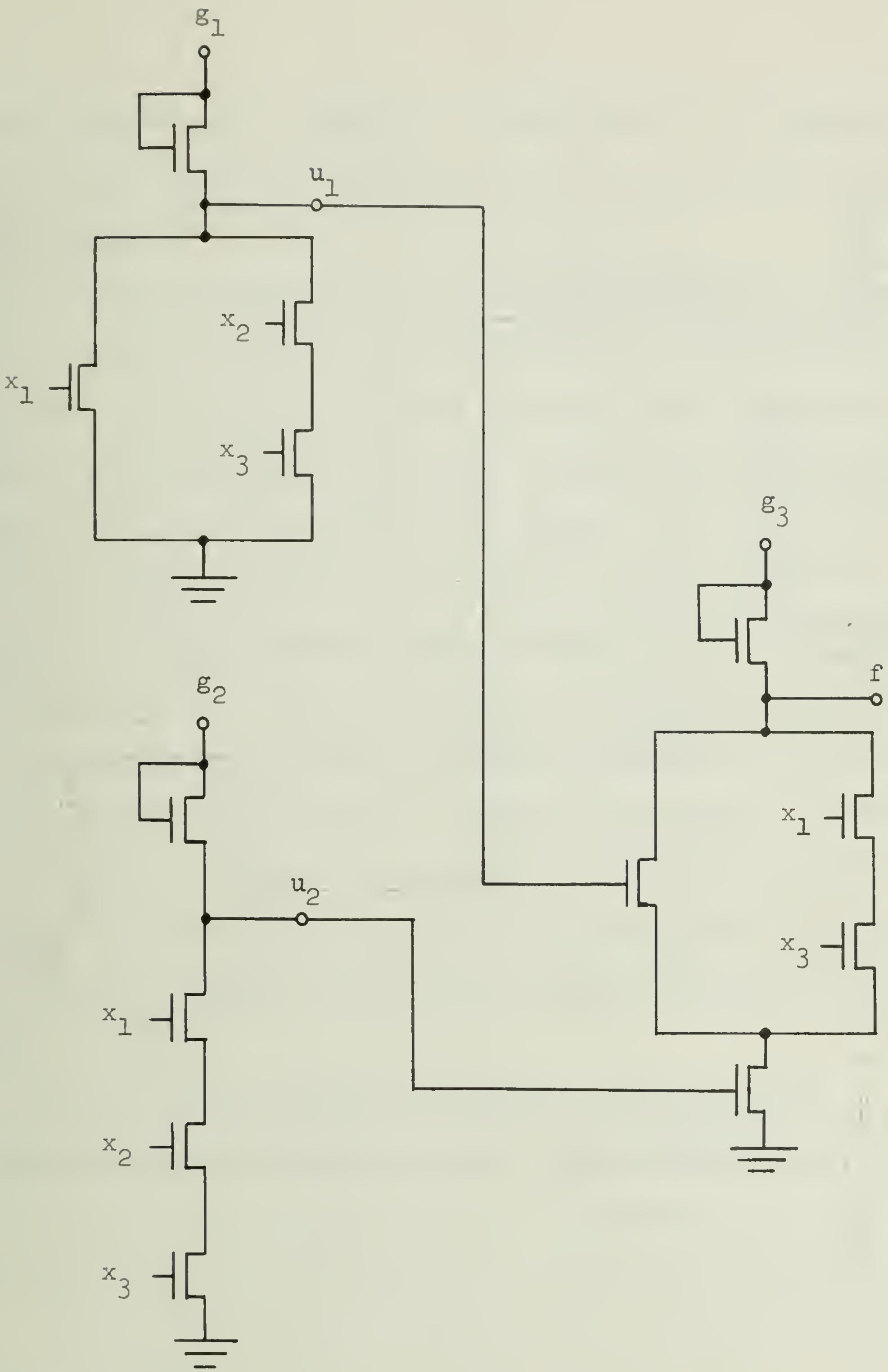


Fig. 4.1.1.3 A 2-level network realizing function f of Fig. 4.1.1.2. Network's synthesis is based on true clusters.

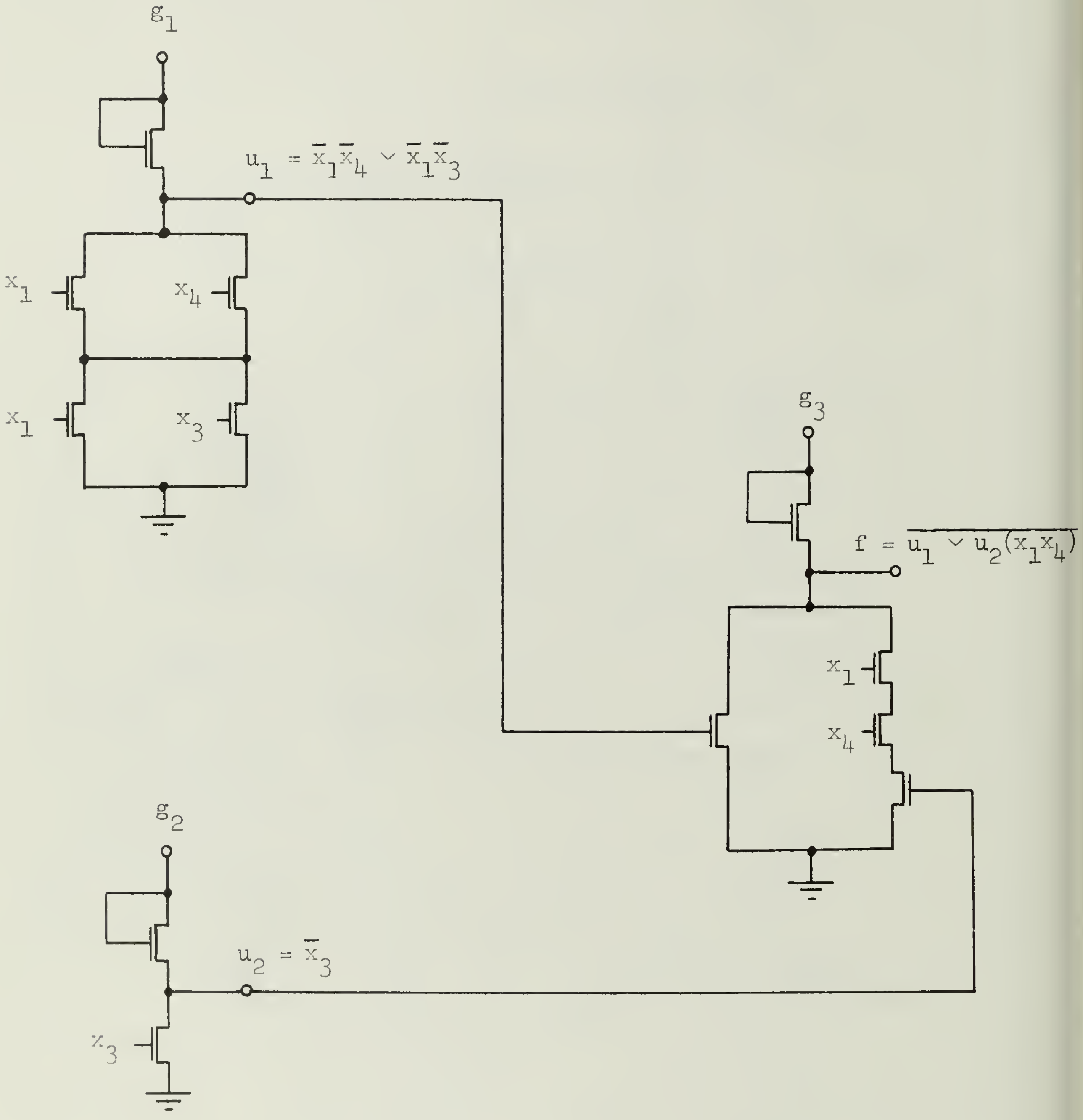

g_1
$u_1 = \bar{x}_1\bar{x}_4 \vee \bar{x}_1\bar{x}_3$
x_1
x_4
x_1
x_3
g_3
$f = \overline{u_1 \vee u_2(x_1x_4)}$
x_1
x_4
g_2
$u_2 = \bar{x}_3$
x_3

Fig. 4.1.1.4 A 2-level network realizing function f of Fig. 4.1.1.2. Network's synthesis is based on false clusters.

in terms of which they are expressed. Based on the false clusters encircled in Fig. 4.1.1.2, the 2-level network of MOS cells shown in Fig. 4.1.1.4 can be constructed.

If one would be able to give a method for selecting, for a given function f, a number of true clusters (or false clusters) such that (i) every true vector (false vector) of f is included in at least one cluster and (ii) the number of these clusters which have floor functions other than the constant 1 (having ceiling functions other than the constant 1) is $D^f(\vec{1},\vec{0})$, then one would have an algorithm for synthesizing G-minimal, 2-level networks of MOS cells (see Theorem 3.8). The approach taken in [Liu 72] and [Liu 75] represents one such method, but others are also possible as will be seen shortly. As a result of the algorithm given in these two papers for choosing true and false clusters, the clusters generated have special relations to each other — relations which are most completely exploited in the related synthesis algorithm for G-minimal, multiple-level networks (given in [Liu 72]). The synthesis algorithms developed by Liu for G-minimal, 2-level networks will be discussed in the next section.

4.1.1.1 Liu's synthesis method for G-minimal, 2-level networks

Liu's method begins with the creation of a 'stratified structure' for a given function f. A stratified structure is essentially a unique partition of all input vectors of the function into true and false clusters, designated M_i^f, i = 0,...,2r, having special properties (from some of which the modifier 'stratified' results).

The following gives Liu's definition of a stratified structure translated into the notation of this paper:

Definition 4.1.1.1 The stratified structure of a function f of n variables is a sequence of subsets of input vectors, $(M_0^f, M_1^f, \ldots, M_{2r}^f)$ where $r = D^f(\vec{1},\vec{0}) + \theta$, $\theta = 0$ if $\vec{0}$ is a false vector and $\theta = 1$ if $\vec{0}$ is a true vector, defined as:

(1) M_{2i+1}^f for $i = 0,1,\ldots,r - 1$ contains every true vector, A, of of f such that $D^f(A,\vec{0}) = i + 1 - \theta$.

(2) M_{2i}^f for $i = 1,2,\ldots,r - 1$ contains every input vector, A, f which satisfies the following conditions:

(i) $A > B$ for at least one $B \in M_{2i-1}^f$;

(ii) $A \not\geq B$ for every $B \in M_{2i+1}^f$;

(iii) $A \notin M_{2i-1}^f$.

(3) M_0^f contains every input vector, A, such that $A \not\geq B$ for every $B \in M_1^f$.

(4) M_{2r}^f contains every input vector, A, such that $A \notin M_{2r-1}^f$ and $A > B$ for at least one $B \in M_{2r-1}^f$.

As previously mentioned, Fig. 4.1.1.1(a) is an example of the stratified structure for function f represented by the labelled n-cube. From top to bottom, the five encircled sets of input vectors are, respectively, M_1^f, M_2^f, M_3^f, M_4^f, and M_5^f. It can be seen that the choice of these M_i^f is the only one which satisfies Definition 4.1.1.1.1.

The following properties of the stratified structure have been shown by Liu [Liu 72] (some aspects are obvious from the definition): Property I (Theorem 2.2.1 and Corollary 2.2.1 of [Liu 72]) $M_0^f, M_1^f, \ldots, M_{2r}^f$ are disjoint; all true vectors are contained in M_{2i+1}^f, $i = 0,1,\ldots,r - 1$, and these subsets contain only true vectors; all false vectors are contained in M_{2i}^f, $i = 0,1,\ldots,r$, and these subsets contain only false vectors; Property II (Theorem 2.2.3 of [Liu 72]) for every two subsets M_i^f and M_j^f, $i < j$, (1) $A \not\geq B$ for every pair of input vectors

$A \in M_i^f$ and $B \in M_j^f$, and (2) for every $B \in M_j^f$ there exists at least one input vector $A \in M_i^f$ such that $B > A$.

Theorem 4.1.1.1.1 Each subset, $M_i^f, i = 0,1,\ldots,2r$, of input vectors in the stratified structure for a function f constitutes either a true cluster or a false cluster.

Proof To be a cluster, either a true cluster or a false cluster, a subset of input vectors, M_i^f, must satisfy the conditions of Definition 4.1.1.2: for every pair of vectors, A_j and A_ℓ, of M_i^f which satisfies $A_j > A_\ell$, every vector A_k satisfying $A_j > A_k > A_\ell$ is also a member of the set. Assume this is not true for some subset M_i^f and input vectors A_j, A_k, A_ℓ. Then $A_k \in M_p^f$ where $p \neq i$. If $p > i$, then $A_j > A_k$ contradicts Property II above. If $i > p$, $A_k > A_\ell$ contradicts Property II. Thus, each M_i^f in the stratified structure of a function f must be a cluster.

Q.E.D.

[Liu 72] proves that the following algorithm generates the stratified structure for a given function f. (This algorithm is slightly different from the form in which it appears in [Liu 72].)

Algorithm 4.1.1.1.1 Algorithm to obtain a stratified structure for a given function f (SS).

Step 1 If $\vec{0}$ is a false vector of f, assign it to M_0^f. Otherwise, assign it to M_1^f. Set $w = 0$.

Step 2 $w = w + 1$.

Step 3 If $w > n$, stop.

Step 4 For each vector A of weight w, let Q_A be the set of vectors B of weight $w - 1$ such that $A > B$, and then assign A to M_t^f where: $t = \max\{p + 1, q\}$, p is the maximum subscript such that M_p^f contains at least one vector in Q_A of the opposite type as A, and q is the maximum subscript such that M_q^f contains at least one vector in Q_A of the same type as A. Go to Step 2.

Upon completion of this algorithm, all vectors in V_n belong to one of the $M_i^f, i = 0,1,\ldots,2r$.

After obtaining the stratified structure for a given function f, the rest of Liu's synthesis method is straightforward. Based on the stratified structure, an expression of f is written (corresponding to either of those given here by Corollary 4.1.1.1) from which the cell configurations of the desired MOS cell network can be directly determined. This expression can be based on either the true or false clusters of the stratified structure, corresponding to Liu's two different synthesis algorithms, respectively.

In [Liu 75] it is mentioned that these synthesis algorithms can be used for incompletely specified functions as well by ignoring unspecified input vectors (only specified true or false vectors are included in the M_i^f). This can be accomplished during the construction of the stratified structure by ignoring unspecified vectors in Step 4 of the preceding algorithm.

In the preceding Fig. 4.1.1.2, the selection of clusters for the function f was exactly that which would have been obtained by Algorithm 4.1.1.1.1 (SS). Following Algorithm SS, all vectors of false cluster I are assigned to M_0^f all vectors of true cluster I are assigned to M_1^f, all vectors of false cluster II are assigned to M_2^f, and all vectors of true cluster II are assigned to M_3^f. This stratified structure leads, of course, to the MOS cell network shown in either Fig. 4.1.1.3 or Fig. 4.1.1.4, depending on whether true or false clusters are chosen to express f, respectively. Thus, these two figures represent the results of Liu's synthesis algorithms for the output function f.

Due to the special nature of the clusters obtained from the stratified structure, the following theorem can be demonstrated.

<u>Theorem 4.1.1.1.2</u> For any two distinct subsets of input vectors, M_i^f and M_j^f, i > j, of the stratified structure, $(M_0^f, M_1^f, \ldots, M_{2r}^f)$, of a function f:

$$\beta_{i,1} \vee \cdots \vee \beta_{i,q_i} \subseteq \beta_{j,1} \vee \cdots \vee \beta_{j,q_j}$$

where $\beta_{i,1} \vee \cdots \vee \beta_{i,q_i}$ and $\beta_{j,1} \vee \cdots \vee \beta_{j,q_j}$ are the floor functions of clusters M_i^f and M_j^f, respectively.

<u>Proof</u> Consider each β-term, $\beta_{i,k}$ of the floor function of M_i^f. Let $B_{i,k}$ be the floor vector corresponding to $\beta_{i,k}$. By Property II of the stratified structure, there must exist a vector $C \in M_j^f$ such that $B_{i,k} > C$. And, by definition of the floor vectors of a cluster, there must exist a floor vector, $B_{j,\ell}$ (with β-term, $\beta_{j,\ell}$), of M_j^f such that $C \geq B_{j,\ell}$. Thus $B_{i,k} > B_{j,\ell}$. Since the β-term of a vector A has the value 1 for an input vector if and only if the input vector is greater than or equal to A, $\beta_{i,k} = 1$ for an input vector only if $\beta_{j,\ell} = 1$. Therefore, for every $\beta_{i,k}$ among $\beta_{i,1}, \ldots, \beta_{i,q_i}$ there exists a $\beta_{j,\ell}$ among $\beta_{j,1}, \ldots, \beta_{j,q_j}$ such that $\beta_{i,k} \subseteq \beta_{j,\ell}$. Hence the disjunction of β-terms corresponding to all floor vectors of M_i^f implies the disjunction of β-terms corresponding to all floor vectors of M_j^f.

Q.E.D.

As a consequence of this theorem, it is also true that the complement of the floor function of M_j^f implies the complement of the floor function of M_i^f, i > j.

Although neither of Liu's synthesis algorithms for 2-level networks utilizes the property of the stratified structure given in Theorem 4.1.1.1.2, this

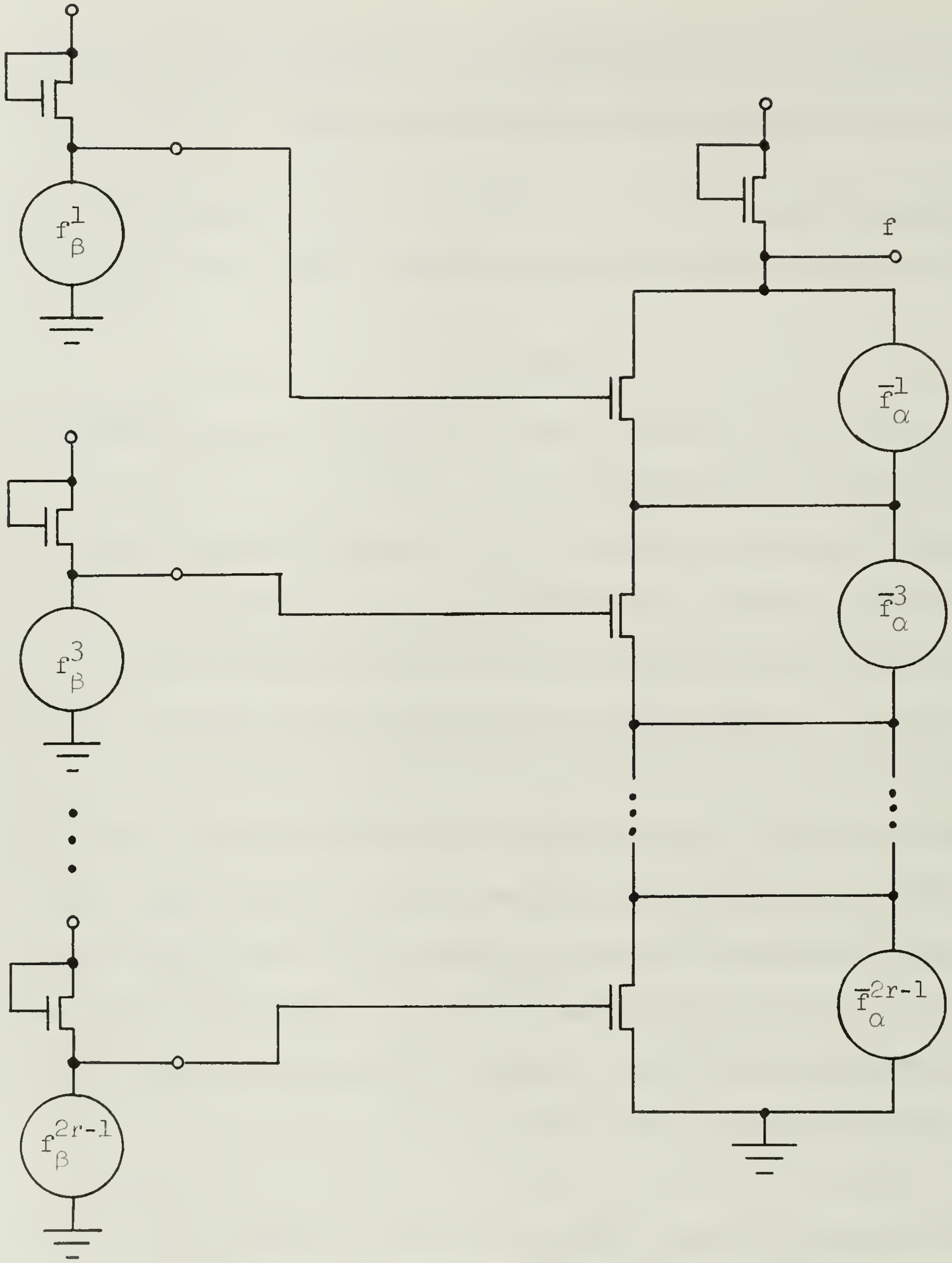


Fig. 4.1.1.1.1 Generalized configuration of a 2-level MOS cell network synthesized by Liu's algorithm based on true vectors.

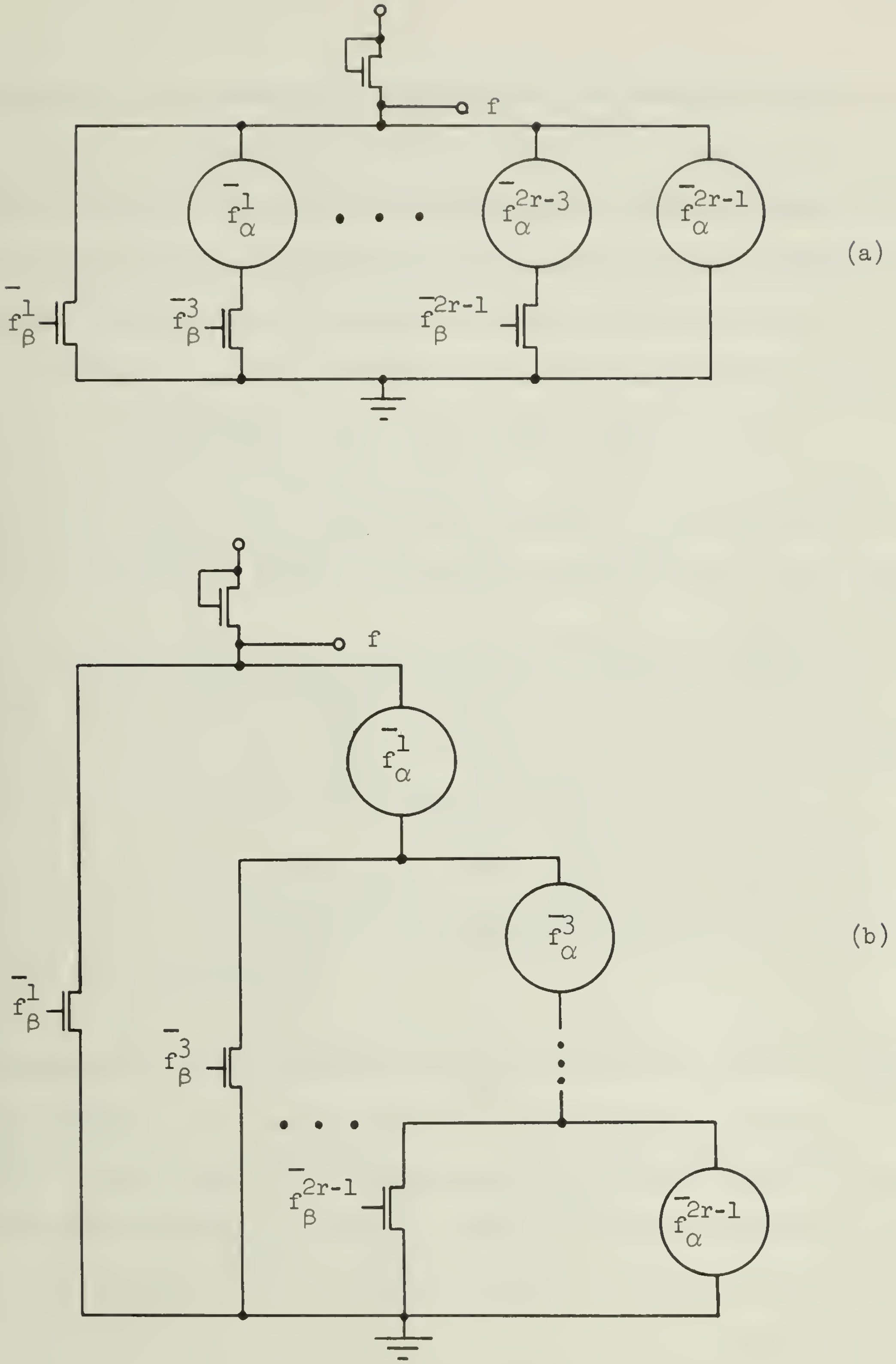


Fig. 4.1.1.1.2 Alternative output cell configurations for the generalized network of Fig. 4.1.1.1.1.

property can be used to obtain alternative and sometimes more desirable configurations of the output cells in the networks synthesized.

For example, suppose the stratified structure for a function f has already been obtained. From this, the following equation based on true clusters, M^f_{2i+1}, $i = 0,1,\ldots,r-1$, can be written (let f^i_α and f^i_β, respectively, represent the ceiling and floor functions of cluster M^f_i) :

$$f = f^1_\alpha f^1_\beta \vee f^3_\alpha f^3_\beta \vee f^5_\alpha f^5_\beta \vee \ldots \vee f^{2r-1}_\alpha f^{2r-1}_\beta .$$

The corresponding MOS cell network configuration is shown, in generalized form (remember that either or both of f^1_β and f^{2r-1}_α could be the constant function 1), in Fig. 4.1.1.1.1.

In this figure, each of the FET's with an input from a second level cell realizes the transmission function $\bar{f}^{2i-1}_\beta$. Based on the fact that $\bar{f}^{2i-1}_\beta \supseteq \bar{f}^{2j-1}_\beta$ for $i > j$, several other configurations of the output cell are also possible. Two of these are shown in Fig. 4.1.1.1.2.

From a practical viewpoint, the configuration of Fig. 4.1.1.1.2(a) may be more desirable than the configuration of the output cell in Fig. 4.1.1.1.1, since it will have, in general, fewer FET's in series (in practice, the number of FET's in series is often more critical than the number in parallel).

It should be noted that while the three output cells shown in these figures realize different functions of their inputs (i.e., if $x_1,\ldots,x_n$, $\bar{f}^1_\beta,\bar{f}^3_\beta,\ldots,\bar{f}^{2r-1}_\beta$ were all independent variables), they all realize function f because of the special relations among the functions $\bar{f}^1_\beta,\bar{f}^3_\beta,\ldots,\bar{f}^{2r-1}_\beta$ realized by the cells of the second level.

Similar alternative configurations of the output cell exist for the synthesis of the networks based on false clusters M^f_{2i}, $i = 0,1,\ldots,r$, of the stratified structure. In this case, the alternative configurations depend on the fact that $f_{\beta}^{2i} \subseteq f_{\beta}^{2j}$ for $i > j$ (Theorem 4.1.1.1.2).

The clusters, true or false, used in Liu's 2-level synthesis methods are, as discussed, both disjoint and 'stratified' (i.e., satisfy the conditions required for a stratified structure). Actually, the clusters from which an expression is derived for a given function f (according to Corollary 4.1.1.1) need be neither. In fact, without these constraints, improved results (i.e., networks with fewer FET's) can often be obtained. Section 4.1.2 will propose new synthesis methods based on non-stratified (and often, non-disjoint) clusters.

The following algorithm[†] to label an n-cube for a function f is given in [Lai 76] as an essential part of Lai's method for synthesizing G-minimal, irredundant (i.e., no FET's may be removed from the network without creating erroneous outputs for some input vectors) networks of MOS cells. By observing the correspondence between Liu's stratified structure and the labelled n-cube resulting from this algorithm, the relation between Liu's stratified structure algorithm and Nakamura et al.'s minimum n-cube labelling algorithm is suggested (as will be seen in Section 4.1.1.2).

Algorithm 4.1.1.1.2 Algorithm based on maximum labelling with a minimum number of bits (MXL).

Let $L_{mx}^{f}(A)$ be the binary label attached to each vertex $A \in C_n$ by this algorithm for a given function f. Let $R_f = \lceil \log_2(D^f(\vec{1},\vec{0}) + 1) \rceil + 1$.

Step 1 Assign $L_{mx}^{f}(\vec{0}) = 2^{R_f} - 2 + f(\vec{0})$. Set $w = 0$.

Step 2 $w = w + 1$.

[†] This algorithm will also be important for the discussion in Section 7.

Step 3 If $w > n$, stop.

Step 4 For each vertex A of weight w, let Q_A be the set of vectors B of weight $w - 1$ such that $A > B$, and then assign as $L^f_{mx}(A)$ the largest binary integer satisfying the two conditions:

(i) The least significant bit of $L^f_{mx}(A)$ is $f(A)$;

(ii) $L^f_{mx}(A) \leq L^f_{mx}(B)$ for every $B \in Q_A$.

Go to Step 2.

In the labelled n-cube resulting from Algorithm MXL, each bit of each binary label is important for (multiple-level) network synthesis as will be discussed later.

It can be demonstrated that each set consisting of all vectors, A, having identical binary labels, $L^f_{mx}(A)$, is a cluster. Furthermore, if $f(\vec{0}) = 1$,

$$\{A | A \in V_n \text{ and } L^f_{mx}(A) = i\} = M^f_{2^{R_f}-i} ,$$

and if $f(\vec{0}) = 0$

$$\{A | A \in V_n \text{ and } L^f_{mx}(A) = i\} = M^f_{2^{R_f}-2-i} .$$

In other words, the results of Algorithm MXL can be interpreted as a partitioning of the input vectors for a given function into a stratified structure.

Fig. 4.1.1.1.3 and Fig. 4.1.1.1.4 allow a comparison of the results of Algorithm 4.1.1.1.1 (SS) and Algorithm MXL for a function f. $D^f(\vec{1},\vec{0}) = 2$ and $R_f = 3$. Since $f(\vec{0}) = 1$, $L^f_{mx}(A) + j = 2^{R_f} = 8$ for every vector $A \in M^f_j$. As can be seen from the figures, grouping the vertices given the same labels by Algorithm MXL results in the stratified structure.

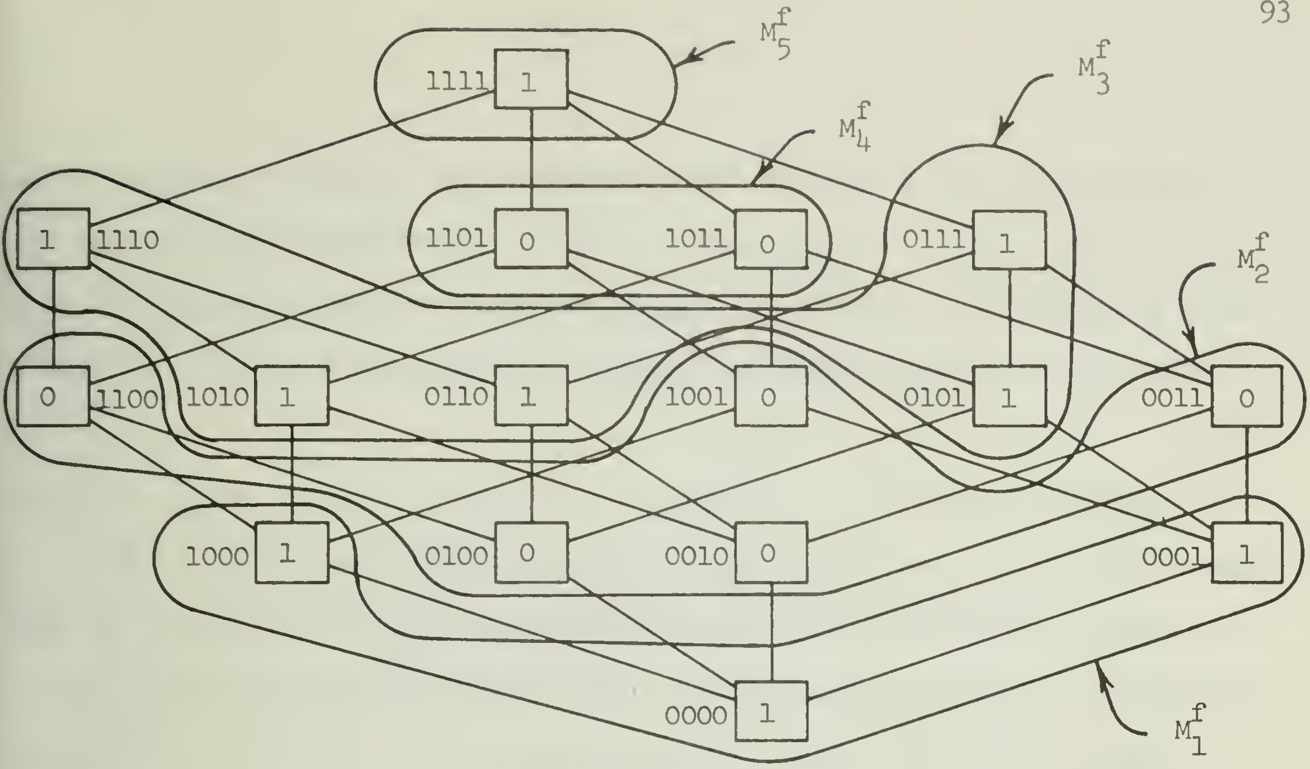


Fig. 4.1.1.1.3 Stratified structure obtained for a function by Algorithm 4.1.1.1.1 (SS).

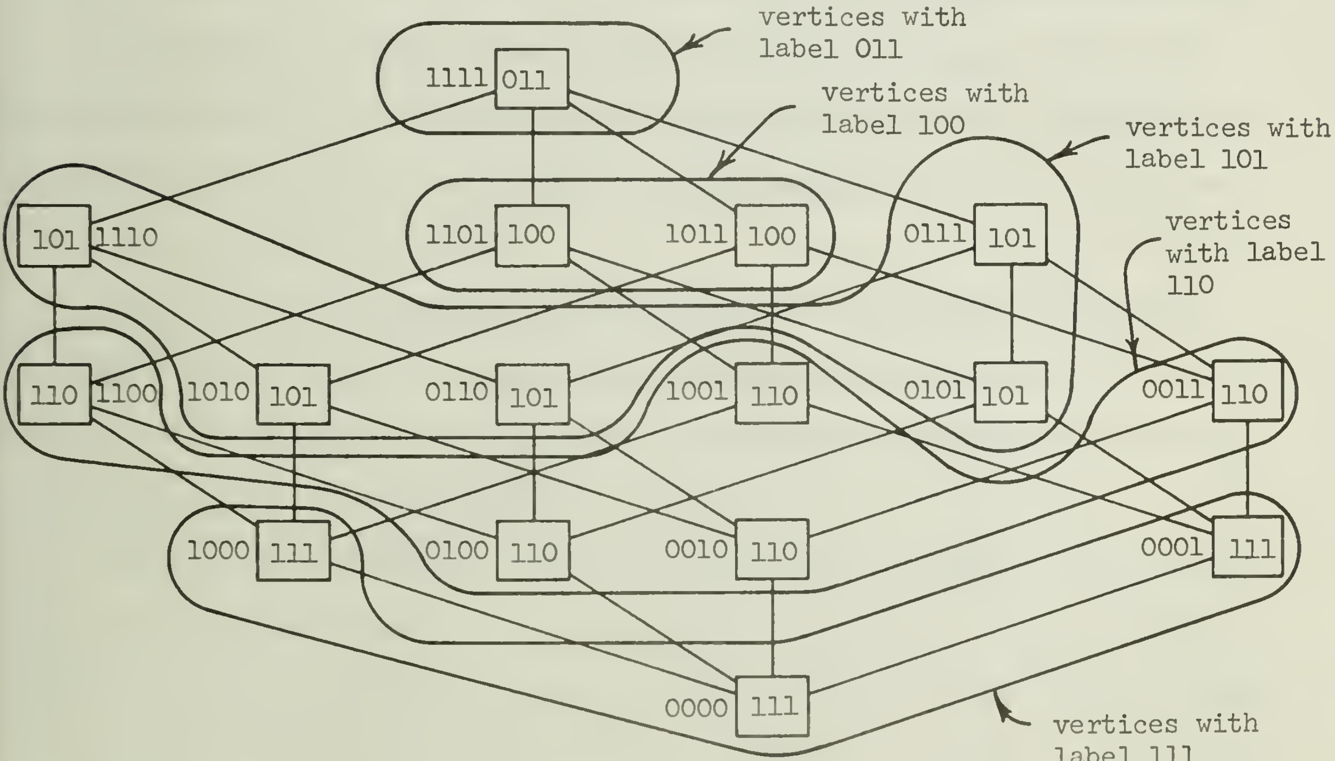


Fig. 4.1.1.1.4 Labelled 4-cube obtained by Algorithm 4.1.1.1.2 (MXL) for same function appearing in Fig. 4.1.1.1.3. Vertices having identical labels are grouped together.

Theorem 4.1.1.1.3 Given a completely specified function f of n variables, for each input vector $A \in V_n$ assigned to M^f_i by Algorithm 4.1.1.1 (SS), the label $L^f_{mx}(A) = 2^{R_f} - 2\theta - i$ is assigned to vertex $A \in C_n$ by Algorithm 4.1.1.1.2 (MXL), where $R^f = \lceil \log_2(D^f(\vec{1},\vec{0}) + 1) \rceil + 1$, $\theta = 0$ if $f(\vec{0}) = 1$, and $\theta = 1$ if $f(\vec{0}) = 0$.

Proof This proof can be accomplished by induction on the weight w of input vectors. For $w = 0$, i.e., for $\vec{0}$, there are two cases:

Case 1 $f(\vec{0}) = 0$. By Step 1 of Algorithm MXL, $L^f_{mx}(\vec{0}) = 2^{R_f} - 2$. By Step 1 of Algorithm SS, $\vec{0}$ is assigned to M^f_0, and the theorem is true in this case.

Case 2 $f(\vec{0}) = 1$. By Step 1 of Algorithm MXL, $L^f_{mx}(\vec{0}) = 2^{R_f} - 1$. By Step 1 of Algorithm SS, $\vec{0}$ is assigned to M^f_1, and, again, the theorem is true.

Now suppose the theorem holds for all vectors of weight w. Let A be any true vector of weight $w + 1$.

Let $\{B_1,\ldots,B_k\}$ be the set of all false vectors such that B_j, $1 \leq j \leq k$, is of weight w and $A > B_j$. Let $\{C_1,\ldots,C_\ell\}$ be the set of all true vectors such that C_j, $1 \leq j \leq \ell$ is of weight w and $A > C_j$. Let $b_j = L^f_{mx}(B_j)$, $j = 1,\ldots,k$, and $c_j = L^f_{mx}(C_j)$, $j = 1,\ldots,\ell$.

By Step 4 of Algorithm MXL, $L^f_{mx}(A)$ is clearly $\min\{b_1 - 1,\ldots, b_k - 1, c_1,\ldots,c_\ell\}$. Since the theorem is true for vectors of weight w, $B_1,\ldots,B_k$, $C_1,\ldots,C_\ell$ are assigned by Algorithm SS to $M^f_{2^{R_f}-2\theta-b_1},\ldots,M^f_{2^{R_f}-2\theta-b_k}$, $M^f_{2^{R_f}-2\theta-c_1},\ldots,M^f_{2^{R_f}-2\theta-c_\ell}$, respectively. Thus, by Step 4 of Algorithm SS, vector A belongs to M^f_i where $i = \max\{2^{R_f} - 2\theta - b_1 + 1,\ldots,2^{R_f} - 2\theta - b_k + 1, 2^{R_f} - 2\theta - c_1,\ldots,2^{R_f} - 2\theta - c_\ell\} = 2^{R_f} - 2\theta + \max\{1 - b_1,\ldots,1 - b_k, - c_1,\ldots, - c_\ell\} = 2^{R_f} - 2\theta - \min\{b_1 - 1,\ldots,b_k - 1, c_1,\ldots,c_\ell\} = 2^{R_f} - 2\theta - L^f_{mx}(A)$. Thus, the theorem is true for every true vector of weight w + 1 if it is true for every vector of weight w.

A similar argument can be made for any false vector A of weight w + 1. Hence, by induction, the theorem statement is valid for every input vector A of f.

Q.E.D.

It is also interesting to note that the results produced by the algorithm for synthesizing G-minimal, 2-level networks in [IM 71] and the results produced by the corresponding algorithm in [Liu 75] or [Liu 72] are very closely related.

This similarity of results arises due to the relation between clusters of the stratified structure of [Liu 75] and compatible sets of essential supplementary columns chosen in [IM 71] from graph G_f developed for a given function f.

Corresponding to the results of Liu's synthesis algorithm based on the true clusters of a stratified structure, second level cells of identical configurations can be obtained by the synthesis algorithm of Ibaraki and Muroga under the following conditions:

(1) When obtaining negative completions for the conjoints of the compatible sets of essential supplementary columns, assign the value 0 for all unspecified vectors greater than existing specified false vectors. Assign the value 1 for all other unspecified vectors.

(2) From these negative completions given in truth table form, develop an expression for the function of each required second level cell based on minimal false vectors (i.e., based on the floor function of the single cluster consisting of all false vectors). The configuration of each second level cell can be obtained directly from the respective expression.

Corresponding to the results of Liu's synthesis algorithm based on the false clusters of a stratified structure, second level cells of identical configuration can be obtained by the synthesis algorithm of Ibaraki and Muroga under the following conditions:

(1) When obtaining negative completions for the conjoints of the compatible sets of essential supplementary columns, assign the value 1 for all unspecified vectors less than existing specified true vectors. Assign the value 0 for all other unspecified vectors.

(2) From these negative completions given in truth table form, develop an expression for the function of each required second level cell based on maximum true vectors (i.e., based on the ceiling function of the single cluster consisting of all true vectors). The configuration of each second level cell can be obtained directly from the respective expression (since all of the variables in the ceiling function are complemented, disjunction and conjunction in the expression correspond to conjunction and disjunction in the configuration, respectively).

Furthermore, after the selection of second level cells realizing functions identical to those realized by second level cells chosen by Liu's algorithm, Ibaraki and Muroga's algorithm can also allow the creation of an output cell identical in configuration to that resulting from Liu's algorithm.

Thus, the networks producible by Liu's 2-level synthesis algorithm are a subset (in general, a proper subset) of those producible by Ibaraki and Muroga's algorithm. Observation of the correspondence between Ibaraki and Muroga's algorithm and the two algorithms of Liu enables a more thorough understanding of the relation between the algorithm based on true clusters and the algorithm based on false clusters of the stratified structure.

4.1.1.2 Nakamura, Tokura, and Kasami's n-cube labelling method as a synthesis method for G-minimal, 2-level networks

In the preceding section (4.1.1.1) the correspondence between the maximum labelling of an n-cube with a minimum number of bits (i.e., the result of Algorithm 4.1.1.1.2 (MXL) and the stratified structure (the result of Algorithm 4.1.1.1.1 (SS)) was demonstrated. There exists a counterpart to the stratified structure defined by Liu which has equivalent properties enabling it to be used in both new 2-level and new multiple-level, G-minimal network synthesis algorithms. This counterpart is related to the n-cube labelling scheme used in [NTK 72] (as a means to synthesize multiple-level, G-minimal networks) in the same manner in which Liu's stratified structure is related to the scheme for maximum labelling, Algorithm MXL. Although the new class of synthesis algorithms thus created produces results of generally the same quality as those of [Liu 72] (since the clusters on which they are based have the same 'horizontal' characteristic as the clusters of Liu's stratified structure), for a given function, the new algorithms may obtain a superior result (and vice versa).

The following algorithm, proposed in [NTK 72], for labelling an n-cube for a function f of n variables will be referred to as the algorithm based on minimum labelling (as named in [Lai 76]).

Algorithm 4.1.1.2.1 Algorithm based on minimum labelling (MNL).

Let $L_{mn}^{f}(A)$ be the binary label attached to each vertex $A \in C_n$ by this algorithm for a given function f.

Step 1 Assign $L_{mn}^{f}(\vec{1}) = f(\vec{1})$. Set $w = n$.

Step 2 $w = w - 1$.

Step 3 If $w < 0$, stop.

Step 4 For each vertex A of weight w, let Q^A be the set of vectors B of weight $w + 1$ such that $B > A$, and then assign as $L^f_{mn}(A)$ the smallest binary integer satisfying the two conditions:

(i) The least significant bit of $L^f_{mn}(A)$ is $f(A)$;

(ii) $L^f_{mn}(A) \geq L^f_{mn}(B)$ for every $B \in Q^A$.

Go to Step 2.

The following theorem was proved in [Lai 76]:

Theorem 4.1.1.2.1 $L^f_{mn}(A)$, the label attached to vertex A by Algorithm 4.1.1.2.1 (MNL) for a function f, has the value $L^f_{mn}(A) = 2D^f(\vec{1},A) + f(A)$ for every $A \in C_n$.

As a consequence of this theorem, the label attached to vertex $\vec{0}$ by Algorithm MNL is $2D^f(\vec{1},\vec{0}) + f(\vec{0})$.

Theorem 4.1.1.2.2 Each set consisting of all vectors $A \in V_n$ corresponding to vertices assigned identical binary labels, $L^f_{mn}(A)$, by Algorithm 4.1.1.2.1 (MNL) for a function f of n variables, is a cluster.

Proof Let Q denote any set consisting of all vectors for which the associated vertices are assigned identical labels by Algorithm MNL for a function f. It is clear by condition (i) of Step 4 of the algorithm that no two vectors of opposite types (i.e., one true vector and one false vector) can be members of the same set Q. Now suppose Q is not a cluster. Then, by the definition of a cluster, there must exist three vectors, $A \in Q$, $B \notin Q$, and $C \in Q$, such that $A > B > C$. By condition (ii) of Step 4, of the algorithm,

$L^f_{mn}(A) \leq L^f_{mn}(B) \leq L^f_{mn}(C)$. Since $L^f_{mn}(A) = L^f_{mn}(C)$, then also $L^f_{mn}(B) = L^f_{mn}(A) = L^f_{mn}(C)$. This implies $B \in Q$ which contradicts the assumption that $B \notin Q$. Thus, no set of vectors, A, B, and C, satisfying the above conditons can exist, and set Q must be a cluster.

Q.E.D.

Clearly $L^f_{mn}(\vec{0}) = 2D^f(\vec{1},\vec{0}) + f(\vec{0})$ is the greatest possible label which can be assigned by Algorithm MNL to a vertex in C_n (if it were not, condition (ii) of Step 4 could be used to show a contradiction). Since the label assigned to the vertex of a true vector always has a '1' as its least significant bit, the maximum possible number of different labels assigned to vertices representing true vectors (i.e., the maximum number of true clusters) is $D^f(\vec{1},\vec{0}) + 1$ if $\vec{0}$ is a true vector and $D^f(\vec{1},\vec{0})$ if $\vec{0}$ is a false vector. In either case, following Corollary 4.1.1.1 it can be seen that a network consisting of $D^f(\vec{1},\vec{0}) + 1$ cells ($D^f(\vec{1},\vec{0})$ second level cells and one output cell) can be constructed. (Recall that if $\vec{0}$ is a true vector, the cluster which includes it will have the constant '1' as a floor function, and no cell corresponding to this cluster is necessary.) By Theorem 3.8, a 2-level network of $D^f(\vec{1},\vec{0}) + 1$ cells is a G-minimal, 2-level network for a given completely specified function f.

In a similar manner, a G-minimal, 2-level network can be constructed based on the false clusters (in the sense of Theorem 4.1.1.2.2) created by Algorithm MNL.

As an example of the n-cube labelling produced by Algorithm MNL and the G-minimal, 2-level networks which can be obtained from it by grouping into clusters all vectors corresponding to vertices assigned the same label, consider the function f shown in Fig. 4.1.1.2.1 (this same function was considered in Fig. 4.1.1.1.3 and Fig. 4.1.1.1.4). The labelled n-cube resulting from Algorithm MNL is given in Fig. 4.1.1.2.2. This figure also shows the grouping of vertices having the same labels (note that the resulting clusters are significantly different from those produced in a similar manner by Algorithm 4.1.1.1.2 (MXL)).

Based on the true clusters in Fig. 4.1.1.2.2, the following expression for f can be obtained according to Corollary 4.1.1.1:

$$f = 1(x_2x_3) \vee (\bar{x}_2\bar{x}_4 \vee \bar{x}_1\bar{x}_3)(x_1 \vee x_4) \vee (\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4)1$$

This, in turn, suggests a G-minimal, 2-level realization consisting of three MOS cells, g_1, g_2, and g_3, realizing the following functions, u_1, u_2, and $u_3 = f$, respectively:

$$u_1 = \overline{x_2x_3}$$

$$u_2 = \overline{x_1 \vee x_4}$$

$$u_3 = \overline{u_1((x_2 \vee x_4)(x_1 \vee x_3) \vee u_2)(x_1 \vee x_2 \vee x_3 \vee x_4)} = f.$$

The corresponding network is shown in Fig. 4.1.1.2.3(a).

Based on the false clusters in Fig. 4.1.1.2.2, the following expression for f can be obtained:

$$f = \overline{(\bar{x}_2 \vee \bar{x}_3)(x_1x_2 \vee x_1x_4 \vee x_3x_4) \vee (\bar{x}_1\bar{x}_3\bar{x}_4 \vee \bar{x}_1\bar{x}_2\bar{x}_4)(x_2 \vee x_3)}$$

The functions, u_1, u_2, and u_3(= f) of the three cells, g_1, g_2, and g_3 in the corresponding G-minimal, 2-level network are:

$$u_1 = \overline{x_2x_3}$$

$$u_2 = \overline{(x_1 \vee x_3 \vee x_4)(x_1 \vee x_2 \vee x_4)}$$

$$u_3 = \overline{u_1(x_1x_2 \vee x_1x_4 \vee x_3x_4) \vee u_2(x_2 \vee x_3)} = f.$$

The corresponding network is shown in Fig. 4.1.1.2.3(b).

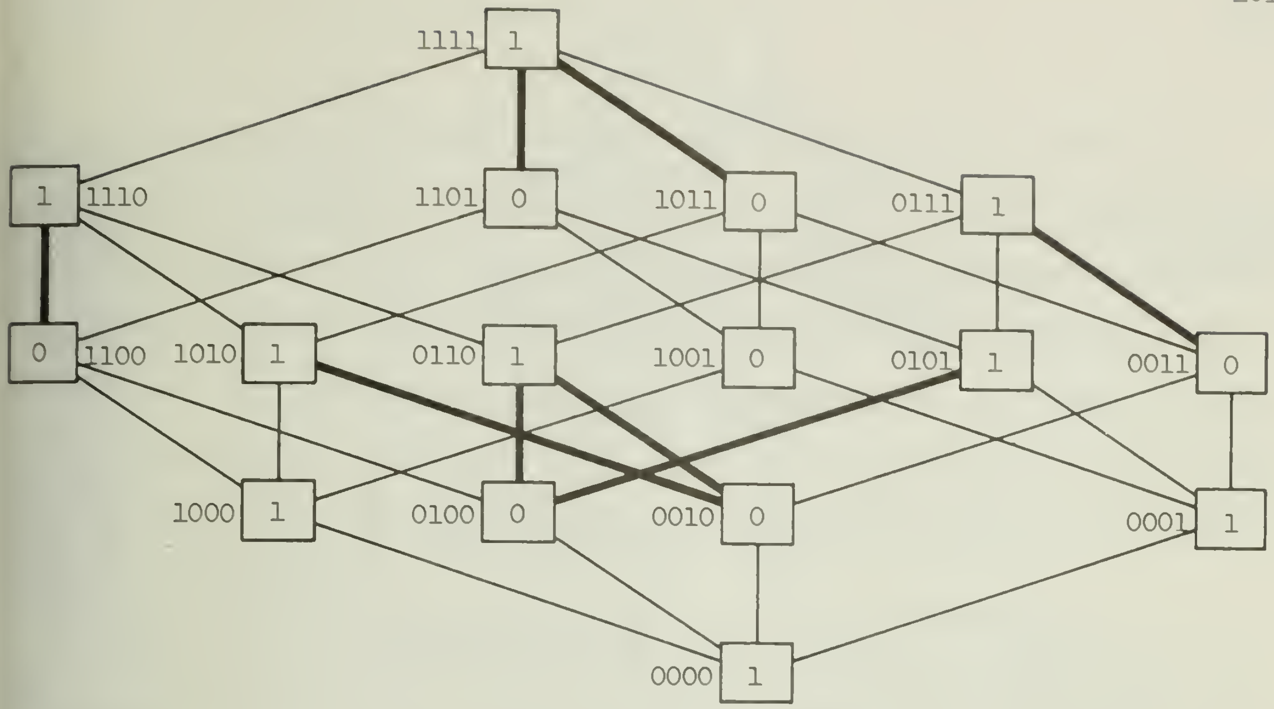


Fig. 4.1.1.2.1 Function for demonstration of Algorithm 4.1.1.2.1 (MNL).

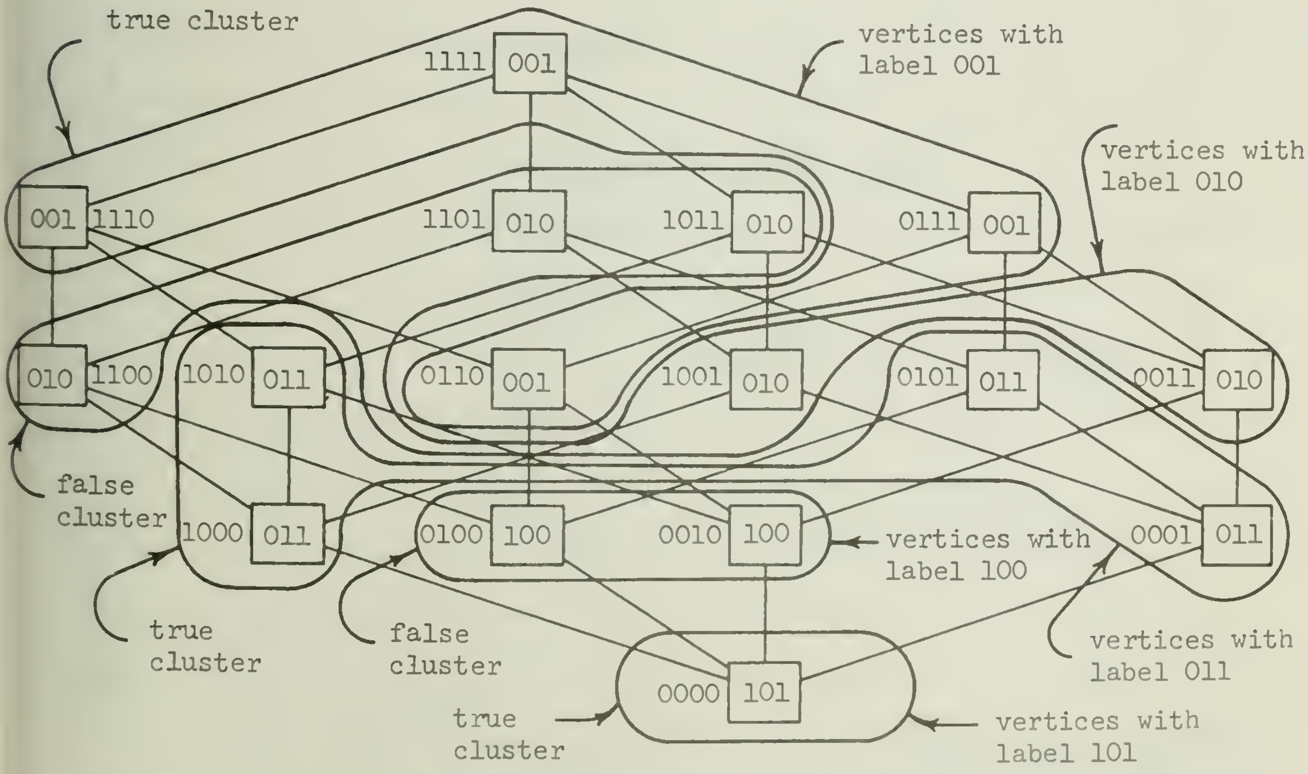
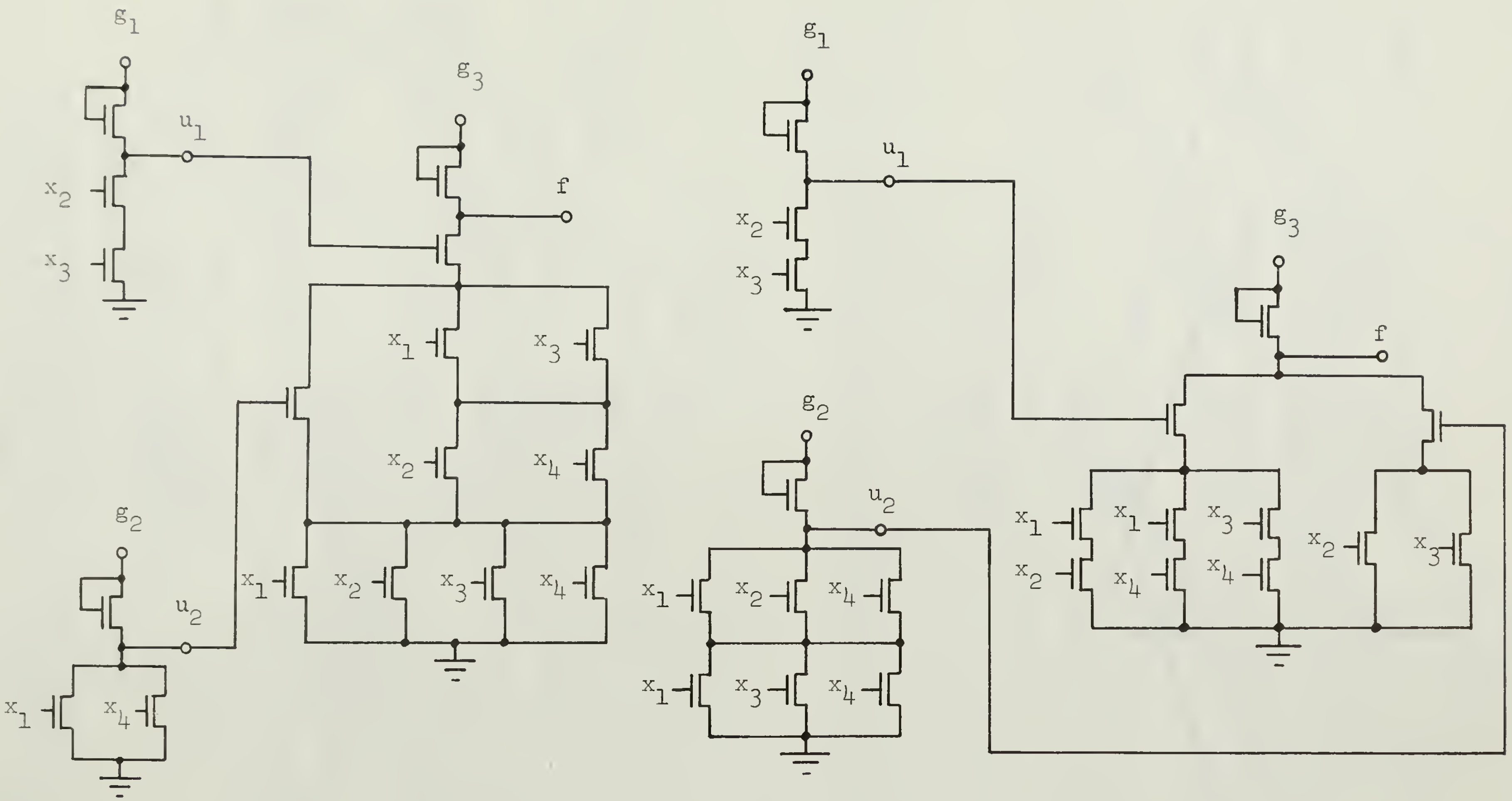


Fig. 4.1.1.2.2 Labelled 4-cube obtained by Algorithm 4.1.1.2.1 (MNL) for function appearing in Fig. 4.1.1.2.1. Vertices having identical labels are grouped together.



(a) Network based on true clusters.

(b) Network based on false clusters.

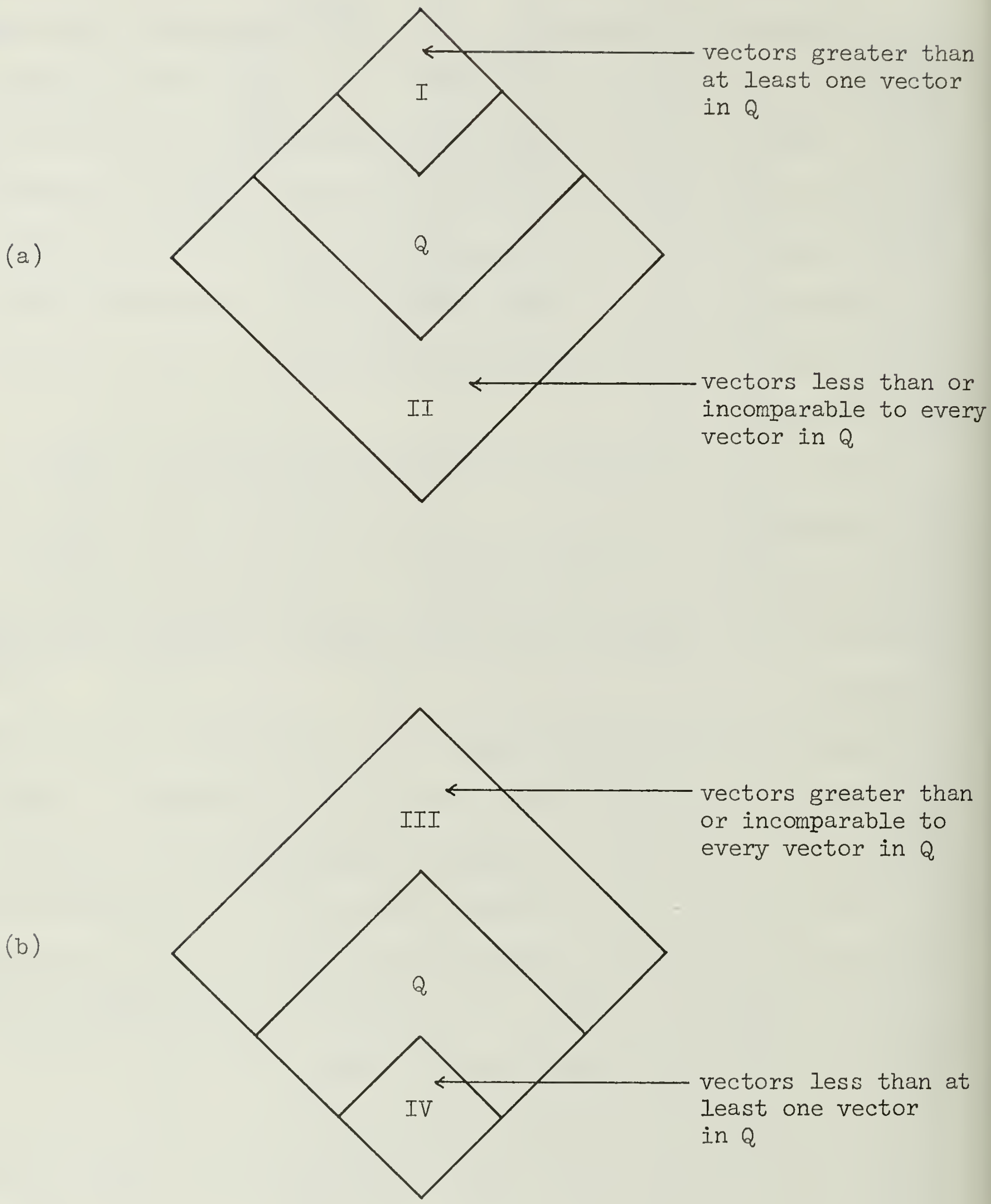
<u>Fig. 4.1.1.2.3</u> G-minimal, 2-level networks for function f (given in Fig. 4.1.1.2.1) obtained from clusters resulting from the grouping of vertices of C_n assigned identical labels by Algorithm 4.1.1.2.1 (MNL) (see Fig. 4.1.1.2.2).

Let pictorial representations for certain relationships among input vectors be introduced in Fig. 4.1.1.2.4. For a given cluster of input vectors, $Q \in V_n$: region I in Fig. 4.1.1.2.4(a) represents all vectors not in Q which are greater than at least one vector in Q; region II in (a) represents all vectors either less than or incomparable to every vector in Q; region III in (b) represents all vectors either greater than or incomparable to every vector in Q; region IV in (b) represents all vectors not in Q which are less than at least one vector in Q.

With this representation, the stratified structure of a function f can be illustrated as in Fig. 4.1.1.2.5. In this figure, the relative positions of the clusters are shown with the same V-shapes as Fig. 4.1.1.2.4(a). This representation is also indicative of the origin of the name 'stratified structure.'

Using the chevron shapes of Fig. 4.1.1.2.4(b), Fig. 4.1.1.2.6 illustrates the relation of the clusters which result from the grouping of vertices of C_n assigned the same label by Algorithm 4.1.1.2.1(MNL) for a function f.

The clusters created by Algorithm MNL (i.e., the n-cube labelling method of [NTK 72]) have properties which correspond to those (e.g., Properties I and II and Theorems 4.1.1.1.1, 4.1.1.1.2, etc.) of Liu's stratified structure (which can be obtained through the use of Algorithm 4.1.1.1.2 (MXL)). It has just been shown that G-minimal, 2-level networks can easily be derived, based on these new clusters, in much the same manner that Liu proposed based on the clusters of his stratified structure. In addition to this, the parallel between the structure consisting of clusters created by Algorithm MNL and the stratified structure will also enable the synthesis of G-minimal, multiple-



vectors greater than at least one vector in Q

vectors less than or incomparable to every vector in Q

vectors greater than or incomparable to every vector in Q

vectors less than at least one vector in Q

Fig. 4.1.1.2.4 Pictorial representation of certain relationships among input vectors.

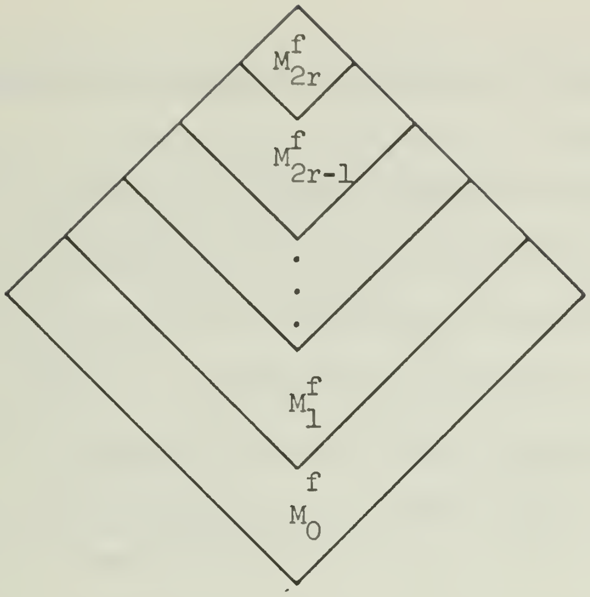


Fig. 4.1.1.2.5 Pictorial representation of a stratified structure (i.e., among clusters created by Algorithm 4.1.1.1.2 (MXL)).

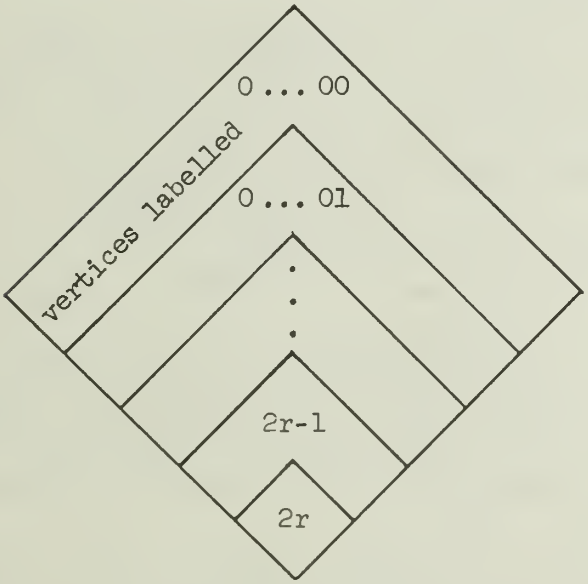


Fig. 4.1.1.2.6 Pictorial representation of relationships among clusters created by Algorithm 4.1.1.2.1 (MNL).

level networks based on the former (in a manner similar to that in which it is accomplished, based on the latter, in [Liu 72]).

For the convenience of future discussion, let Liu's stratified structure for a function f (Definition 4.1.1.1.1) be referred to as the <u>BU-stratified structure</u> (Bottom Up) for a function f. Let the structure represented by the clusters obtained, as described, through the use of Algorithm MNL be called the <u>TD-stratified structure</u> (Top Down) for a function f.

Consider the following structure consisting of sets of input vectors for a given function.

<u>Structure I</u> Let $(W_0^f, W_1^f, \ldots, W_{2r}^f)$ be the sequence of subsets of input vectors defined as follows for a function f of n variables, where $r = D^f(\vec{1}, \vec{0}) + \theta$, $\theta = 0$ if $\vec{0}$ is a false vector and $\theta = 1$ if $\vec{0}$ is a true vector:

(1) W_{2i+1}^f for $i = 0,1,\ldots,r - 1$ contains every true vector, A, of f such that $D^f(\vec{1}, A) = i$.

(2) W_{2i}^f for $i = 1,2,\ldots,r - 1$ contains every input vector, A, of f which satisfies the following conditions:

(i) $A < B$ for at least one $B \in W_{2i-1}^f$;

(ii) $A \not\leq B$ for every $B \in W_{2i+1}^f$;

(iii) $A \in W_{2i+1}^f$.

(3) W_0^f contains every input vector, A, such that $A \not\leq B$ for every $B \in W_1^f$.

(4) W_{2r}^f contains every input vector, A, such that $A \notin W_{2r-1}^f$ and $A < B$ for at least one $B \in W_{2r-1}^f$.

It can be shown that the structure $(W_0^f, W_1^f, \ldots, W_{2r}^f)$ defined above is actually the TD-stratified structure obtained by Algorithm 4.1.1.2.1 (MNL) (i.e., grouping vertices of the same labels assigned by Algorithm MNL produces $W_0^f, W_1^f, \ldots, W_2^f$). Furthermore, the TD-stratified structure can be demonstrated to have properties exactly paralleling those of the BU-stratified structure.

4.1.2 Synthesis methods for 2-level networks based on non-stratified structures

The true and false clusters of the stratified structure which form the basis of the synthesis algorithms presented in Section 4.1.1.1 and 4.1.1.2 possess special properties which are actually unnecessary in the synthesis of G-minimal, 2-level networks. For example, the clusters of both the BU- and TD-stratified structures are disjoint and they have the 'stratified' relations pictured in Fig. 4.1.1.2.5 and 4.1.1.2.6. Furthermore, in order to produce these properties, the characteristics of the individual clusters are generally not the most desirable (compared with other selections of sets of clusters which are not necessarily stratified)for obtaining a G-minimal, 2-level network having a relatively small number of FET's.

A typical cluster in a stratified structure tends to be rather 'horizontal' in shape when viewed in the context of an n-cube (i.e., there is usually not a wide variation among weights of the different vectors in the same cluster). T This 'horizontal shape' generally results in relatively large numbers of floor and ceiling vectors. These, in turn, lead to large numbers of literals in the corresponding floor and ceiling functions, and these literals have a 1-to-1 correspondence with the number of FET's in the resulting MOS cell network.

Of course, the expressions (see Corollary 4.1.1.1) corresponding to the MOS cell configurations can be factored to obtain cells of fewer FET's. However, the large number of original literals increases the factorization problem. Furthermore, if clusters with simpler floor and ceiling functions are selected, factoring the simpler expressions can often lead to a final result with still fewer FET's.

To produce a G-minimal, 2-level network for a function f based on true or false clusters, Q_i, $i = 1,\ldots,r$, as outlined near the end of the introduction to Section 4.1.1, only the condition on the Q_i given in Corollary 4.1.1.1 need be met for $r = D^f(\vec{1},\vec{0}) + \theta$. Clusters need be neither disjoint nor stratified nor have the property that every pair of vectors, A and B, in the same cluster satisfies $D^f(\vec{1},A) = D^f(\vec{1},B)$ or $D^f(A,\vec{0}) = D^f(B,\vec{0})$ – these being characteristics of the clusters of the TD- and BU-stratified structures. Free of these restrictions, clusters can be selected for use which are more 'vertical' in shape (i.e., have relatively fewer floor and ceiling vectors for a given number of vectors in the cluster) than those resulting from either Algorithm 4.1.1.2.1 (MNL) or Algorithm 4.1.1.1.2 (MXL). As discussed, a synthesis based on such clusters seems likely to lead to 2-level MOS networks having relatively small numbers of FET's. The desired algorithm, based on the new concept of clusters introduced in Definition 4.1.1.2, can be stated generally as follows:

Algorithm 4.1.2.1 General algorithm, based on a non-stratified structure, to synthesize a G-minimal, 2-level network of MOS cells implementable with a small number of FET's for a given function f of n variables (GNS).

Step 1 Determine the inversion degree, $D^f(\vec{1},\vec{0})$, of function f.

Step 2 Decide whether the synthesis will be based on true clusters or false clusters of f.

Step 3 Determine the number of true clusters (false clusters, if so decided in Step 2), r, which must be chosen to yield a G-minimal network:

$$r = D^f(\vec{1},\vec{0}) + \theta,$$

where: $\theta = 1$ if $\vec{0}$ is a true vector ($\vec{1}$ is a false vector); $\theta = 0$ if $\vec{0}$ is a false vector ($\vec{1}$ is a true vector).

Step 4 Select r true clusters (false clusters), $Q_1, \ldots, Q_r$, such that $Q_1 \cup \ldots \cup Q_r$ is the set of all true vectors (false vectors) of f. Attempt to choose the Q_i such that: the numbers of ceiling and floor vectors are as small as possible; the weights of the ceiling vectors are as large as possible; and the weights of the floor vectors are as small as possible.

Step 5 Express f ($\bar{f}$) in terms of the ceiling and floor functions of Q_i:

$$(\alpha_{1,1} \vee \ldots \vee \alpha_{1,p_1})(\beta_{1,1} \vee \ldots \vee \beta_{1,q_1}) \vee \;.\;\;.\;\;.\; \vee$$

$$(\alpha_{r,1} \vee \ldots \vee \alpha_{r,p_r})(\beta_{r,1} \vee \ldots \vee \beta_{r,q_r}) = f (= \bar{f}),$$

where: p_i and q_i are, respectively, the numbers of ceiling vectors and floor vectors of cluster Q_i; $\alpha_{i,1} \vee \ldots \vee \alpha_{i,p_i}$ and $\beta_{i,1} \vee \ldots \vee \beta_{i,q_i}$ are, respectively, the ceiling and floor functions for cluster Q_i.

Step 6 Define u_i as the complement of the floor function (ceiling function) for true cluster (false cluster) Q_i. Express $u_1, \ldots, u_r$, and f as follows:

$$u_1 = \overline{\beta_{1,1} \vee \ldots \vee \beta_{1,q_1}}$$

$$\vdots \qquad\qquad \vdots$$

$$u_r = \overline{\beta_{r,1} \vee \ldots \vee \beta_{r,q_r}}$$

$$f = \overline{(\bar{\alpha}_{1,1} \ldots \bar{\alpha}_{1,p_1} \vee u_1) \;\;.\;\;.\;\;.\;\; (\bar{\alpha}_{r,1} \ldots \bar{\alpha}_{r,p_r} \vee u_r)}$$

$$(u_1 = \overline{\bar{\alpha}_{1,1} \ldots \bar{\alpha}_{1,p_1}}$$

$$\vdots \qquad\qquad \vdots$$

$$u_r = \overline{\bar{\alpha}_{r,1} \ldots \bar{\alpha}_{r,q_r}}$$

$$f = \overline{u_1(\beta_{1,1} \vee \ldots \vee \beta_{1,q_1}) \vee \;.\;\;.\;\;.\; \vee u_r(\beta_{r,1} \vee \ldots \vee \beta_{r,q_r})}\;)$$

Step 7 The configuration of a G-minimal, 2-level network consisting of $r + 1 - \theta$ MOS cells can be obtained directly from the $r + 1$ expressions resulting from Step 6. Each expression defines the output function and configuration of a corresponding cell of the network, with the possible exception of one expression for a u_i, $1 \leq i \leq r$, which, when $\theta = 1$, is the constant 0 (constant 1, if synthesis is based on false clusters) and requires no corresponding cell.

Example 4.1.2.1 Consider once more the function f of Fig. 4.1.1.2. Let Algorithm GNS be employed for this function.

In Step 1, it is determined that $D^f(\vec{1},\vec{0}) = 2$ (consider the directed path (111) → (101) → (100) → (000)). Suppose it is decided in Step 2 that the synthesis will be based on true clusters. By Step 3, the number of clusters to be chosen, r, is two. The following two true clusters are selected in accordance with Step 4 (see Fig. 4.1.2.1):

$$Q_1 = \{(111),(011)\},$$

$$Q_2 = \{(110),(100)\}.$$

Following Step 5, f can be expressed as:

$$1(x_2x_3) \vee \bar{x}_3x_1 = f.$$

Step 6 gives expressions which correspond to the output functions and configurations of individual cells in the network being synthesized:

$$u_1 = \overline{x_2x_3}$$

$$u_2 = \overline{x_1}$$

$$f = \overline{u_1(x_3 \vee u_2)}$$

By Step 7, the MOS cell network is constructed as shown in Fig. 4.1.2.2. Compare this result with that in Fig. 4.1.1.3 obtained based on the true clusters of a BU-stratified structure for f. The number of driver FET's has been reduced from ten to six while the optimum number of cells has been maintained. Results obtained by the other three methods based on stratified structures would be as follows: synthesis based on false vectors of BU-stratified structure, nine FET's (reducible to eight FET's if expressions of Step 6 are factored); synthesis based on true vectors of TD-stratified structure, ten FET's; synthesis based on false vectors of TD-stratified structure, nine FET's (reducible to eight FET's if expressions of Step 6 are factored).

Example 4.1.2.2 As a second example of the improved results obtainable through the use of Algorithm GNS, consider the function f shown in Fig. 4.1.2.3 with its BU-stratified structure.

In Step 1 of the algorithm, it is determined that $D^f(\vec{1},\vec{0}) = 2$. Again, suppose it is decided in Step 2 to base the synthesis on true clusters. In Step 3, $\theta = 1$ since $\vec{0}$ is a true vector, and the number of true clusters to be chosen is consequently determined to be three (i.e., $r = 3$). The following three true clusters can be selected in accordance with Step 4 (see Fig. 4.1.2.4):

$$Q_1 = \{(1111),(1110),(1011),(0111),(1010),(0110),(0011),(0010)\}$$

$$Q_2 = \{(1110),(1100),(1010),(1000)\}$$

$$Q_3 = \{(1010),(1000),(0010),(0000)\}.$$

As evident from Fig. 4.1.2.4, the chosen true clusters are neither disjoint nor 'stratified.' Following Step 5, f can be expressed as :

$$1(x_3) \vee (\bar{x}_4)(x_1) \vee (\bar{x}_2\bar{x}_4)1 = f.$$

From Step 6, the following expressions are obtained which correspond to the output functions and configurations of individual cells in the network being synthesized:

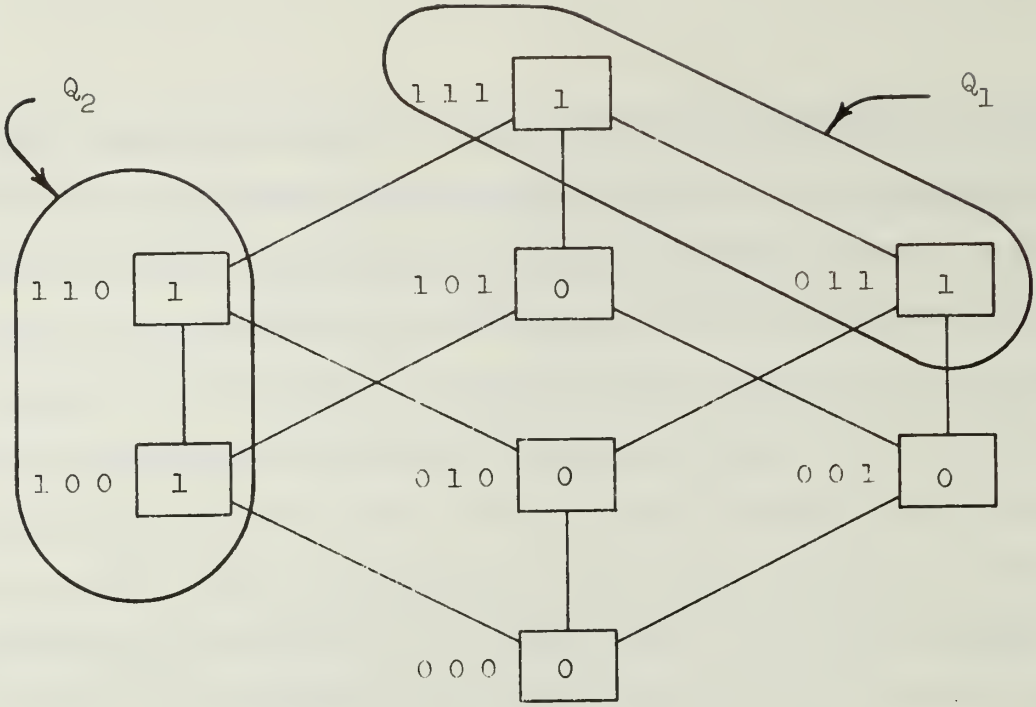


Fig.4.1.2.1 Function f of Fig. 4.1.1.2 with chosen true clusters of non-stratified structure encircled. (Example 4.1.2.1.)

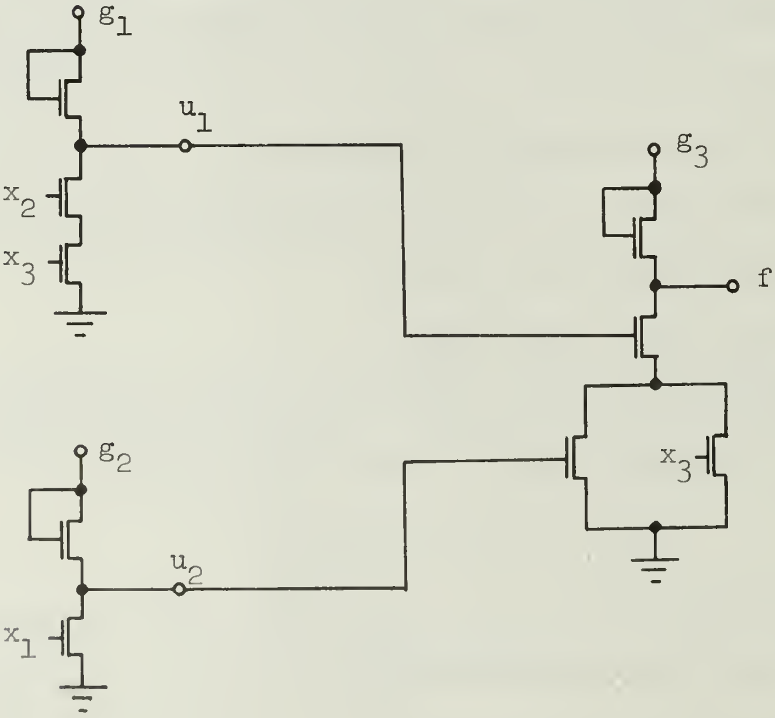


Fig. 4.1.2.2 G-minimal network resulting from Algorithm 4.1.2.1 (GNS) which realizes function f of Fig. 4.1.1.2 and Fig. 4.1.2.1. (Example 4.1.2.1.)

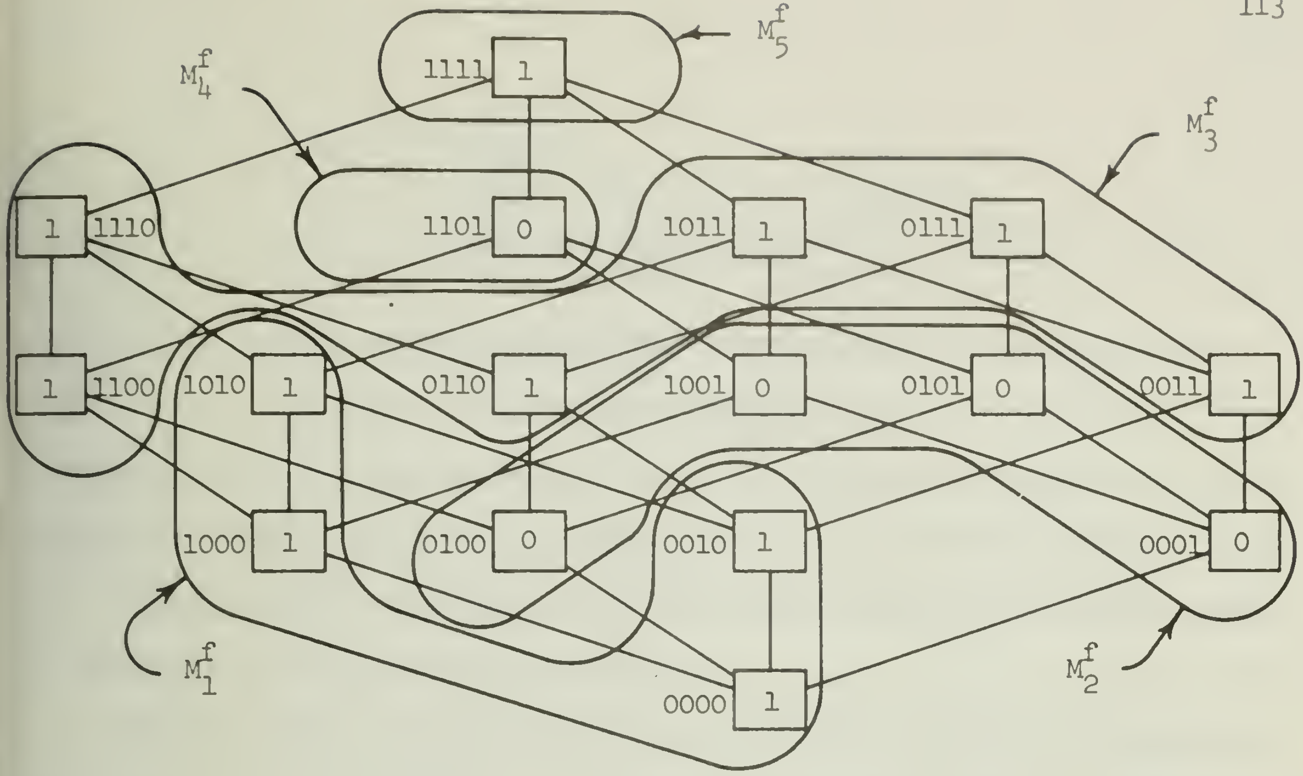


Fig. 4.1.2.3 Given function f with corresponding BU-stratified structure. (Example 4.1.2.2.)

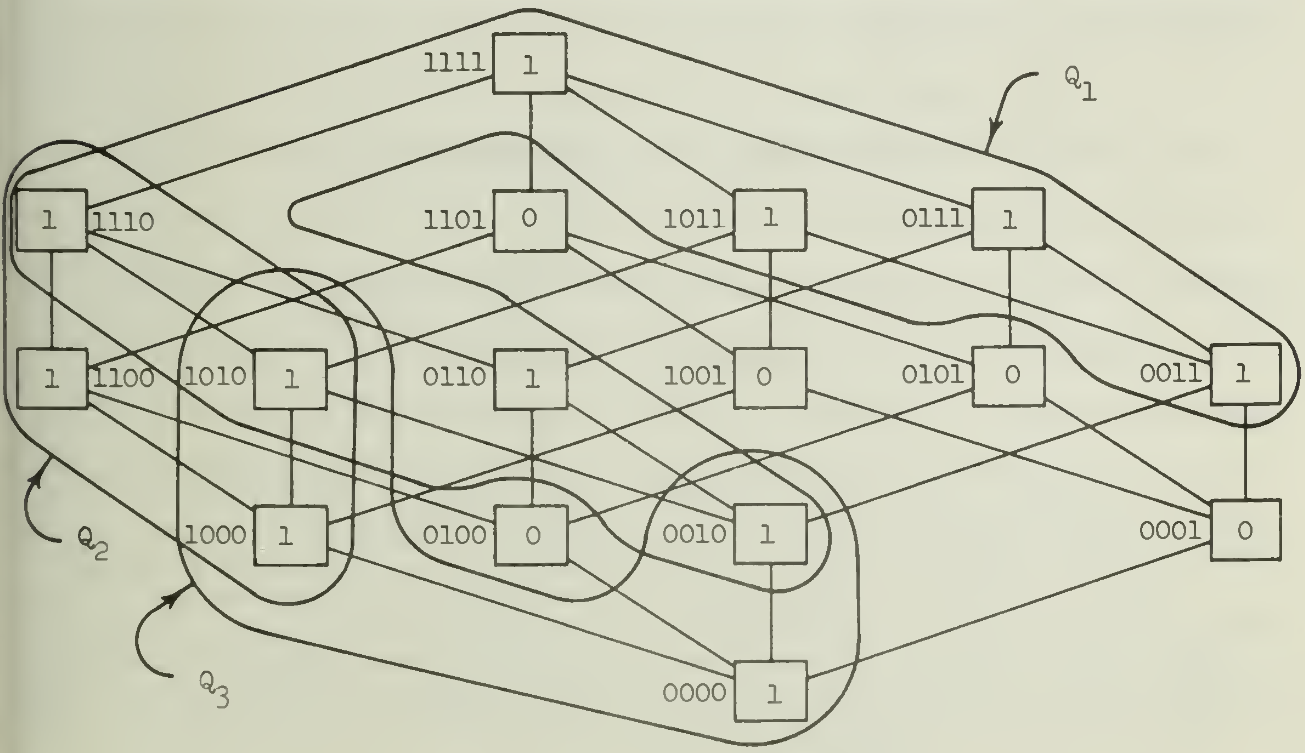


Fig. 4.1.2.4 Function f of Fig. 4.1.2.3 with chosen true clusters of non-stratified structure encircled. (Example 4.1.2.2.)

$$\begin{aligned}
u_1 &= \overline{x_3} \\
u_2 &= \overline{x_1} \\
u_3 &= \overline{1} = 0 \\
f &= \overline{(0 \vee u_1)(x_4 \vee u_2)((x_2 \vee x_4) \vee u_3)} \\
&= \overline{u_1(x_4 \vee u_2)(x_2 \vee x_4)}.
\end{aligned}$$

By Step 7, the MOS cell network can be constructed as shown in Fig. 4.1.2.5. Note the absence of an MOS cell corresponding to the expression for u_3. This network consists of three MOS cells and seven driver FET's. The obvious simplification can be made in the output cell (corresponding to the factored expression: $f = \overline{u_1(u_2x_2 \vee x_4)}$) to reduce this to a total of only six driver FET's.

For comparison, let a second network be constructed which is based on the true clusters of the BU-stratified structure for f. There are three such clusters (see Fig. 4.1.2.3):

$$\begin{aligned}
M_1^f &= \{(1010),(1000),(0010),(0000)\} \\
M_3^f &= \{(1110),(1011),(0111),(1100),(0110),(0011)\} \\
M_5^f &= \{(1111)\}.
\end{aligned}$$

Employing the floor and ceiling functions of these clusters, the function f can be expressed as follows (in accordance with the algorithm given in [Liu 72], [Liu 75]):

$$f = (\overline{x}_2\overline{x}_4)1 \vee (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4)(x_1x_2 \vee x_2x_3 \vee x_3x_4) \vee 1(x_1x_2x_3x_4).$$

Still following the algorithm of [Liu 72], [Liu 75], variables u_1 and u_2 are introduced such that f can be expressed as a negative function of $x_1,\ldots,x_n$, u_1, u_2, and both u_1 and u_2 can be expressed as a negative function of $x_1,\ldots,x_n$:

$$u_1 = \overline{x_1x_2 \vee x_2x_3 \vee x_3x_4}$$

$$
\begin{aligned}
u_2 &= \overline{x_1x_2x_3x_4} \\
f &= (\bar{x}_2\bar{x}_4) \vee (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_4)\bar{u}_1 \vee \bar{u}_2 \\
&= \overline{(x_2 \vee x_4)(x_1x_2x_4 \vee u_1)u_2}.
\end{aligned}
$$

The MOS cell network corresponding to these expressions can be seen in Fig. Fig. 4.1.2.6. The network employs three MOS cells and 17 driver FET's. The configuration of the cell corresponding to the expression for u_1 can be obviously simplified to reduce the total number of required driver FET's to 16. Thus the network of Fig. 4.1.2.5 represents a 59% decrease in the number of driver FET's used to realize function f.

For this problem, a synthesis based on the false clusters of the BU-stratified structure in Fig. 4.1.2.3 would have yielded a network of three cells and 12 driver FET's (11, if simplified). Employing Algorithm GNS in a synthesis based on false clusters would have yielded a network of three cells and seven driver FET's (though having a different configuration than the network shown in Fig. 4.1.2.5).

Algorithm GNS did not suggest how a minimum number of clusters Q_i, upon which the syntheses are based, could be selected from the set of all possible input vectors. Although it is a trivial task to group true or false vectors into subsets Q_i satisfying Corollary 4.1.1.1, it is not a trivial task to insure that the number of such Q_i is minimal. Furthermore, it seems unlikely that any procedure suitable for hand computation can be given which directly (i.e., without 'branching' or 'backtracking') obtains a minimal number of such sets Q_i while secondarily minimizing the total number of literals in the expressions of their corresponding floor and ceiling functions (which have a direct correlation with the number of FET's in the implemented network) for every given function. This objective can be realized, however, in computer

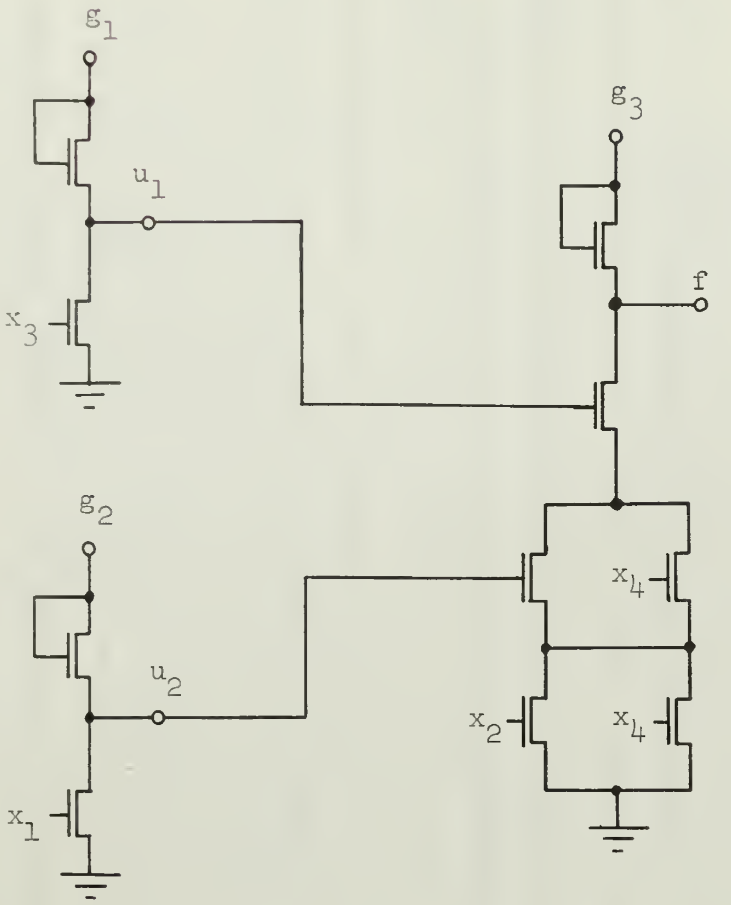


Fig. 4.1.2.5 G-minimal network resulting from Algorithm 4.1.2.1 (GNS) and realizing function f of Fig. 4.1.2.3. (Example 4.1.2.2.)

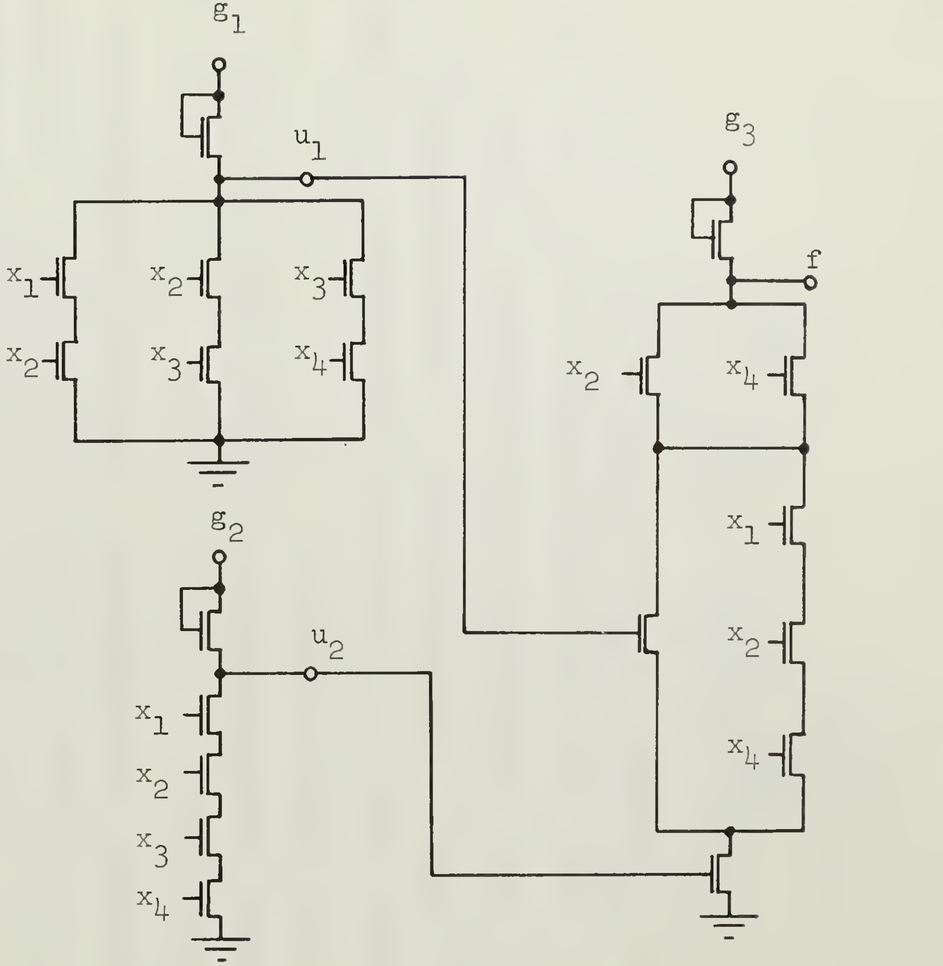


Fig. 4.1.2.6 G-minimal network derived based on true vectors of BU-stratified structure for function f of Fig. 4.1.2.3. (Example 4.1.2.2.)

program implementations of the algorithms (e.g., through the use of integer programming).[†] Hence, to maintain generality in Algorithm GNS, no specific procedure was mentioned for this portion of the algorithm.

The following Algorithm 4.1.2.2 (SNS) for synthesizing G-minimal, 2-level networks of MOS cells does include a specific procedure for choosing a minimum number of sets Q_i which have relatively few literals in the expressions of their corresponding floor and ceiling functions. (It can be shown that a synthesized network always involves a number of FET's smaller than or equal to that of a network synthesized by the corresponding algorithm in [Liu 72] or [Liu 75] for the same function.) As mentioned, an algorithm such as this, i.e., without a 'backtracking' capability, cannot be expected to give G-minimal solutions with a minimum number of FET's as often as an algorithm having this capability.

Although heuristic or enumerative methods could achieve better results in the selection of the Q_i, the use of these more complex algorithms might tend to nullify the main advantages of the synthesis algorithms based on non-stratified (and also stratified) structures: simplicity and feasibility of hand calculation.

Although the following Algorithm 4.1.2.2 (SNS) may seem at first relatively complicated, it is fairly simple in principle and, with use, can soon become easy to apply by hand without the need to refer to the detailed steps. This algorithm employs the BU-stratified structure (a similar algorithm can easily

[†] Even if such a procedure is given, it would not necessarily lead to an algorithm to synthesize 2-level, GF-minimal networks since:

(1) The type of synthesis algorithms being discussed here, i.e., those based on Corollary 4.1.1.1, can produce only a subset of the possible 2-level, G-minimal networks.

(2) Even a selection of Q_i minimizing the number of literals in the floor and ceiling functions before factoring does not always correspond to the minimum number of literals possible after factoring.

be created employing the TD-stratified structure). Clusters, either true or false, of the BU-stratified structure are altered to produce floor and ceiling functions of fewer literals than in the original clusters. This can be seen to lead to synthesized networks of fewer FET's.

Algorithm 4.1.2.2 Specific algorithm, based on a non-stratified structure, to synthesize a G-minimal, 2-level network of MOS cells for a given function f of n variables (SNS).

Step 1 Obtain the BU-stratified structure, $(M_0^f, M_1^f, \ldots, M_{2r}^f)$, for function f by the use of Algorithm 4.1.1.1.1 (SS).

Step 2 Decide whether the synthesis will be based on true clusters or false clusters.

Step 3 If the synthesis is to be based on true clusters, set $i = 2r + 1$. If it is to be based on false clusters, set $i = 2r + 2$. Initially, sets $Q_0, Q_1, \ldots, Q_{2r}$ are empty.

Step 4 $i = i - 2$. If $i < 0$, go to Step 8.

Step 5 Let $Q_i = M_i^f - (Q_{i+2} \cup Q_{i+4} \cup \ldots \cup Q_{2r-t})$ where $t = 1$ if the synthesis is to be based on true clusters and $t = 0$ if it is to be based on false clusters.

Step 6 Attempt to extend the 'ceiling' of cluster Q_i. If the set of ceiling vectors of Q_i is a subset of the set of ceiling vectors of M_i^f, go to Step 7. Otherwise:

(a) Attempt to select a ceiling vector, A, of Q_i which is not a ceiling vector of M_i^f.

(b) If no such vector A exists, go to Step 7.

(c) Select a ceiling vector, B, of M_i^f such that $B > A$. (Such a vector will always exist.)

(d) Add to set Q_i every vector $C \notin Q_i$ such that $B \geq C > D$ for at least one floor vector D of Q_i. Return to Step 6a.

Step 7 Attempt to extend the 'floor' of cluster Q_i.

(a) Seek a vector, $A \notin Q_i$, satisfying:

(i) there exists a floor vector B of Q_i such that $A < B$, and

(ii) $Q_i \cup \{C | C \notin Q_i$ and $A \leq C < D$ for at least one ceiling vector D of $Q_i\}$ is a cluster.

(b) If no such vector A exists, go to Step 4.

(c) Otherwise, add to set Q_i every vector $C \notin Q_i$ such that $A \leq C < D$ for at least one ceiling vector D of Q_i. Return to Step 7a.

Step 8 Let $Q_{j_1},\ldots,Q_{j_s}$ be the resulting non-empty sets among $Q_0,Q_1,\ldots,Q_{2r}$. Express f in terms of the ceiling and floor functions of the Q_{j_i}:

$$\begin{aligned}&(\alpha_{1,1} \vee \cdots \vee \alpha_{1,p_1})(\beta_{1,1} \vee \cdots \vee \beta_{1,q_1}) \vee \quad \cdot \quad \cdot \quad \cdot \quad \vee \\ &(\alpha_{s,1} \vee \cdots \vee \alpha_{s,p_s})(\beta_{s,1} \vee \cdots \vee \beta_{s,q_s}) = \begin{cases} f \text{ if synthesis based on true clusters} \\ \bar{f} \text{ if synthesis based on false clusters}\end{cases}\end{aligned}$$

where: p_i and q_i are, respectively, the number of ceiling vectors and floor vectors of cluster Q_{j_i}; $\alpha_{i,1} \vee \cdots \vee \alpha_{i,p_i}$ and $\beta_{i,1} \vee \cdots \vee \beta_{i,q_i}$ are, respectively, the ceiling and floor functions for cluster Q_{j_i}.

Step 9 Define u_i as the complement of the floor function for cluster Q_{j_i} if the synthesis is being based on true clusters. If the synthesis is being based on false clusters, define u_i as the complement of the ceiling function for cluster Q_{j_i}. Express $u_1,\ldots,u_s$, and f as follows: If the synthesis is being based on true clusters:

$$u_1 = \overline{\beta_{1,1} \vee \cdots \vee \beta_{1,q_1}}$$

$$\vdots \qquad\qquad \vdots$$

$$u_s = \overline{\beta_{s,1} \vee \cdots \vee \beta_{s,q_s}}$$

$$f = \overline{(\bar{\alpha}_{1,1} \cdots \bar{\alpha}_{1,p_1} \vee u_1) \cdots (\bar{\alpha}_{s,1} \cdots \bar{\alpha}_{s,p_s} \vee u_r)};$$

If the synthesis is being based on false clusters:

$$u_1 = \overline{\bar{\alpha}_{1,1} \cdots \bar{\alpha}_{1,p_1}}$$

$$\vdots \qquad\qquad \vdots$$

$$u_s = \overline{\bar{\alpha}_{s,1} \cdots \bar{\alpha}_{s,q_s}}$$

$$f = \overline{u_1(\beta_{1,1} \vee \cdots \vee \beta_{1,q_1}) \vee \cdots \vee u_s(\beta_{s,1} \vee \cdots \vee \beta_{s,q_s})}.$$

<u>Step 10</u> Let $\theta = 1$ if either: the synthesis is being based on true clusters and $\vec{0}$ is a true vector; or the synthesis is being based on false clusters and $\vec{1}$ is a false vector. Let $\theta = 0$ otherwise. Then, the configuration of a G-minimal, 2-level network consisting of $s + 1 - \theta$ MOS cells can be obtained directly from the $s + 1$ expressions resulting from Step 9. Each expression defines the output function and configuration of a corresponding cell of the network, with the possible exception of one expression for a u_i, $1 \le i \le s$, which, when $\theta = 1$, is the constant 0 or 1 and requires no corresponding cell.

This algorithm was designed to synthesize G-minimal networks of a smaller or equal number of FET's than networks synthesized by corresponding algorithms (i.e., based on true vectors or based on false vectors) in [Liu 72] or [Liu 75]. Steps 6 and 7 insure that each chosen cluster has at most as many, and often fewer, total literals in the expressions of its floor and ceiling functions as the respective cluster of the BU-stratified structure. In turn, this means the synthesis of a G-minimal, 2-level network with generally fewer FET's.

Example 4.1.2.3 Consider once again the function of Fig. 4.1.2.3 (Example 4.1.2.2). Step 1 of Algorithm 4.1.2.2 (SNS) gives the BU-stratified structure (note, 2r = 6):

$$M_0^f = \emptyset$$

$$M_1^f = \{(1010),(1000),(0010),(0000)\}$$

$$M_2^f = \{(1001),(0101),(0100),(0001)\}$$

$$M_3^f = \{(1110),(1011),(0111),(1100),(0110),(0011)\}$$

$$M_4^f = \{(1101)\}$$

$$M_5^f = \{(1111)\}$$

$$M_6^f = \emptyset$$

Suppose in Step 2 it is decided to base the synthesis on true clusters. Thus, by Step 3, i = 7. Note that sets Q_1, Q_3, and Q_5 are empty at this point (see row 1 of Table 4.1.2.1). By Step 4, i = 5. In Step 5, $Q_5 = M_5^f = \{(1111)\}$ (row 2 of Table 4.1.2.1). Step 6 is skipped since the ceiling vector of Q_5 is the ceiling vector of M_5^f. In Step 7a, selecting (0010) as vector $A \notin Q_5$ satisfies both conditions i) and ii). Step 7c adds vectors (1110), (1011), (0111), (1010), (0110), (0011), and (0010) to set Q_5 (row 3 of Table 4.1.2.1). Returning to Step 7a, no new vector $A \notin Q_5$ can be found satisfying the conditions. Returning to Step 4, i = 3. By Step 5, $Q_3 = M_3^f - Q_5 = \{(1100)\}$(row 4 of Table 4.1.2.1). In Step 6a, vector A = (1100) is found to be a ceiling vector of Q_3 and not of M_3^f. In Step 6c, only ceiling vector B = (1110) is found to have the relation $B > A$. Step 6d adds vector (1110) to set Q_3 (row 5 of Table 4.1.2.1). Returning to Step 6a, the only ceiling vector, (1110), of Q_3 is also

			$M_5^f =$	$M_3^f =$	$M_1^f =$
			$\{(\overline{\underline{1111}})\}$†	$\{(\overline{1110}),(\overline{1011}),(\overline{0111})\}$ $(\underline{1100}),(\underline{0110}),(\underline{0011})\}$	$\{(\overline{1010}),(1000),$ $(0010),(\underline{0000})\}$
Row	Step	Repetition	$Q_5 =$	$Q_3 =$	$Q_1 =$
1	3	1-st	$\emptyset$	$\emptyset$	$\emptyset$
2	5	1-st	$\{(\overline{\underline{1111}})\}$	$\emptyset$	$\emptyset$
3	7c	1-st	$\{(\overline{1111}),(1110),(1011),(0111),$ $(1010),(0110),(0011),(\underline{0010})\}$	$\emptyset$	$\emptyset$
4	5	2-nd	$\{(\overline{1111}),(1110),(1011),(0111),$ $(1010),(0110),(0011),(\underline{0010})\}$	$\{(\overline{\underline{1100}})\}$	$\emptyset$
5	6d	2-nd	$\{(\overline{1111}),(1110),(1011),(0111),$ $(1010),(0110),(0011),(\underline{0010})\}$	$\{(\overline{1110}),(\underline{1100})\}$	$\emptyset$
6	7c	2-nd	$\{(\overline{1111}),(1110),(1011),(0111),$ $(1010),(0110),(0011),(\underline{0010})\}$	$\{(\overline{1110}),(1100),(1010),(\underline{1000})\}$	$\emptyset$
7	5	3-rd	$\{(\overline{1111}),(1110),(1011),(0111),$ $(1010),(0110),(0011),(\underline{0010})\}$	$\{(\overline{1110}),(1100),(1010),(\underline{1000}\}\}$	$\{(\overline{\underline{0000}})\}$
8	6d	3-rd	$\{(\overline{1111}),(1110),(1011),(0111),$ $(1010),(0110),(0011),(\underline{0010})\}$	$\{(\overline{1110}),(1100),(1010),(\underline{1000})\}$	$\{(\overline{1010}),(1000),(0010),(\underline{0000})\}$

†A bar above or below a vector indicates it is a ceiling or floor vector, respectively, of the M_i^f or C_i.

Table 4.1.2.1 Example 4.1.2.3: Sets Q_5, Q_3, Q_1 following steps of Algorithm 4.1.2.2 (SNS).

a ceiling vector of M_3^f, so nothing more can be added to Q_3 in Step 6. In Step 7 vector A = (1000) satisfies the two conditions of Step 7a. In Step 7c, vectors (1010) and (1000) are added to set Q_3 (row 6 of Table 4.1.2.1). Returning to Step 7a, no new vector $A \notin Q_3$ can be found which satisfies the conditions. Returning to Step 4, i = 1. By Step 5, $Q_1 = M_1^f - (Q_3 \cup Q_5) = \{(0000)\}$ (row 7 of Table 4.1.2.1). In Step 6a, vector A = (0000), a ceiling vector for Q_1, is found not to be a ceiling vector of M_1^f. Since only one vector, (1010), of M_1^f is a ceiling vector, it is selected as vector B in Step 6c. By Step 6d, vectors (1010), (1000), and (0010) are added to set Q_1 (row 8 of Table 4.1.2.1). Returning to Step 6a, no additional vector A can be selected. In Step 7a, no vector A exists satisfying the conditions. Returning to Step 4, i = -1. By Step 8, Q_1, Q_3, and Q_5 are the non-empty sets, and they lead to the following expression of f (refer to last row of Table 4.1.2.1):

$$f = (\bar{x}_2\bar{x}_4)1 \vee (\bar{x}_4)(x_1) \vee 1(x_3).$$

By Step 9:

$$\begin{aligned}
u_1 &= \bar{1} = 0\\
u_2 &= \bar{x}_1\\
u_3 &= \bar{x}_3\\
f &= \overline{((x_2 \vee x_4) \vee u_1)(x_4 \vee u_2)(0 \vee u_3)}\\
&= \overline{(x_2 \vee x_4)(x_4 \vee u_2)u_3}.
\end{aligned}$$

By Step 10, the corresponding network can be determined (see Fig. 4.1.2.5). By coincidence, the synthesized network is equivalent to that previously derived in Example 4.1.2.2 through the use of Algorithm 4.1.2.1 (GNS). The chosen true clusters are identical in both cases; however, this will not be true in general. It was already seen that this network is greatly improved over that derived based on true vectors of the BU-stratified structure (see Fig. 4.1.2.6).

4.2 Synthesis of G-Minimal, Multiple-Level Networks

If a network to realize a given function or set of functions is not restricted to two levels, even fewer negative gates are required, in general (see Theorems 3.8 and 3.9). Several new synthesis algorithms will be proposed in this section which produce G-minimal, multiple-level MOS networks for given functions.

Section 4.2.1 will first discuss the synthesis algorithm of [Liu 72] for G-minimal, single-output, multiple level networks of MOS cells. Liu's algorithm is based on the floor functions of both true and false clusters (M_i) composing the BU-stratified structure of a given function. Also in Section 4.2.1, it will be demonstrated that a corresponding synthesis algorithm can be developed based on ceiling functions of the clusters (W_i^f) of TD-stratified structures. Generally this new algorithm should produce results of comparable quality (in terms of numbers of FET's used) to those of the algorithms of [Liu 72], but for particular given functions the algorithm based on the TD-stratified structure can produce a better result (and vice versa for other functions).

Additional new algorithms will then be discussed in Section 4.2.2 which can produce improved results (again, in terms of numbers of FET's). These algorithms first determine the (BU- or TD-) stratified structure of a given function. The clusters $M_0^f, M_1^f, \ldots, M_{2r}^f$ or $W_0^f, W_1^f, \ldots, W_{2r}^f$ form the basis for the selection of a new set of clusters called 'extended M_i^f' or 'extended W_i^f', respectively. Based upon the floor functions of the extended M_i^f or the ceiling functions of the extended W_i^f, configurations of the cells in a G-minimal MOS network realizing the given function can be easily determined. The new algorithms obtain networks of fewer FET's than the corresponding algorithms of Section 4.2.1, while maintaining the ease of derivation of the cell configurations which is characteristic of the latter.

4.2.1 Synthesis methods for multiple-level networks based on stratified structures and corresponding floor or ceiling functions

The approach taken in [Liu 72] to the G-minimal synthesis problem is based on the same BU-stratified structure used in the synthesis algorithms for the 2-level-restriced case. In the multiple-level case, however, the synthesis involves an additional labelling procedure. Instead of assigning a label to each vertex of an n-cube as in Algorithm 4.1.2.1.2 (MXL) or Algorithm 4.1.2.2.1 (MNL), this labelling assigns a label to each cluster M_i^f of the stratified structure (all vertices involved in a single cluster implicitly receive the same label). Generally several different such labellings (corresponding to different network configurations) will exist, for a given function, which satisfy required conditions.

Liu's synthesis method has the advantage that, once the BU-stratified structure and labelling have been determined,† the detailed configuration of the corresponding network of MOS cells can almost immediately be obtained. At this point in other algorithms, significantly (in a relative sense) more effort is needed, in general, before arriving at the configurations of the MOS cells (the extent of this effort being dependent on the amount of acceptable redundancy). Furthermore, [Liu 72] shows (Theorems 4.2.1 and 4.2.2 of [Liu 72]) that only the special type of labelled n-cubes mentioned above (i.e., all vertices in the same M_i^f receiving the same label) need be considered when either a G-minimal or GI-minimal network is desired.

†Actually, the floor functions of the M_i^f are also needed, but these can be obtained quite easily during the determination of the BU-stratified structure as a by-product of the calculation.

For the development of new MOS network synthesis algorithms in this and the following section (4.2.2), Liu's synthesis algorithm will be presented after first giving certain necessary preliminary definitions (much of the terminology and notation is as given in [Liu 72]).

The following definition (using a notation developed in [Lai 76]) will also be important for the discussion of Section 7.

Definition 4.2.1.1 A negative function sequence of length R for function f of n variables, denoted by $NFS_n(R,f)$ is an ordered set of R functions in C_n (i.e., functions of $x_1,\ldots,x_n$), $u_1,\ldots,u_R$, such that:

(i) u_1 is a negative function of $x_1,\ldots,x_n$;

(ii) u_i is a negative function with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$ for $i = 2,\ldots,R$; and

(iii) $u_R = f$.

It can easily be seen that the $NFS_n(R,f)$ represents the functions realized by the negative gates in the generalized form of a feed-forward network of R negative gates given in Fig. 2.11. Gate g_i in Fig. 2.11 realizes function u_i (or any negative completion) of $NFS_n(R,f)$. Due to this correspondence, the problem of finding a G-minimal negative gate network is equivalent to the problem of finding an $NFS_n(R,f) = (u_1,\ldots,u_R = f)$ such that R is minimized. Let the notation R_f be used to denote the number of negative gates in a G-minimal network for a given function f and let $NFS_n(R_f,f)$ denote a minimum negative function sequence for f.

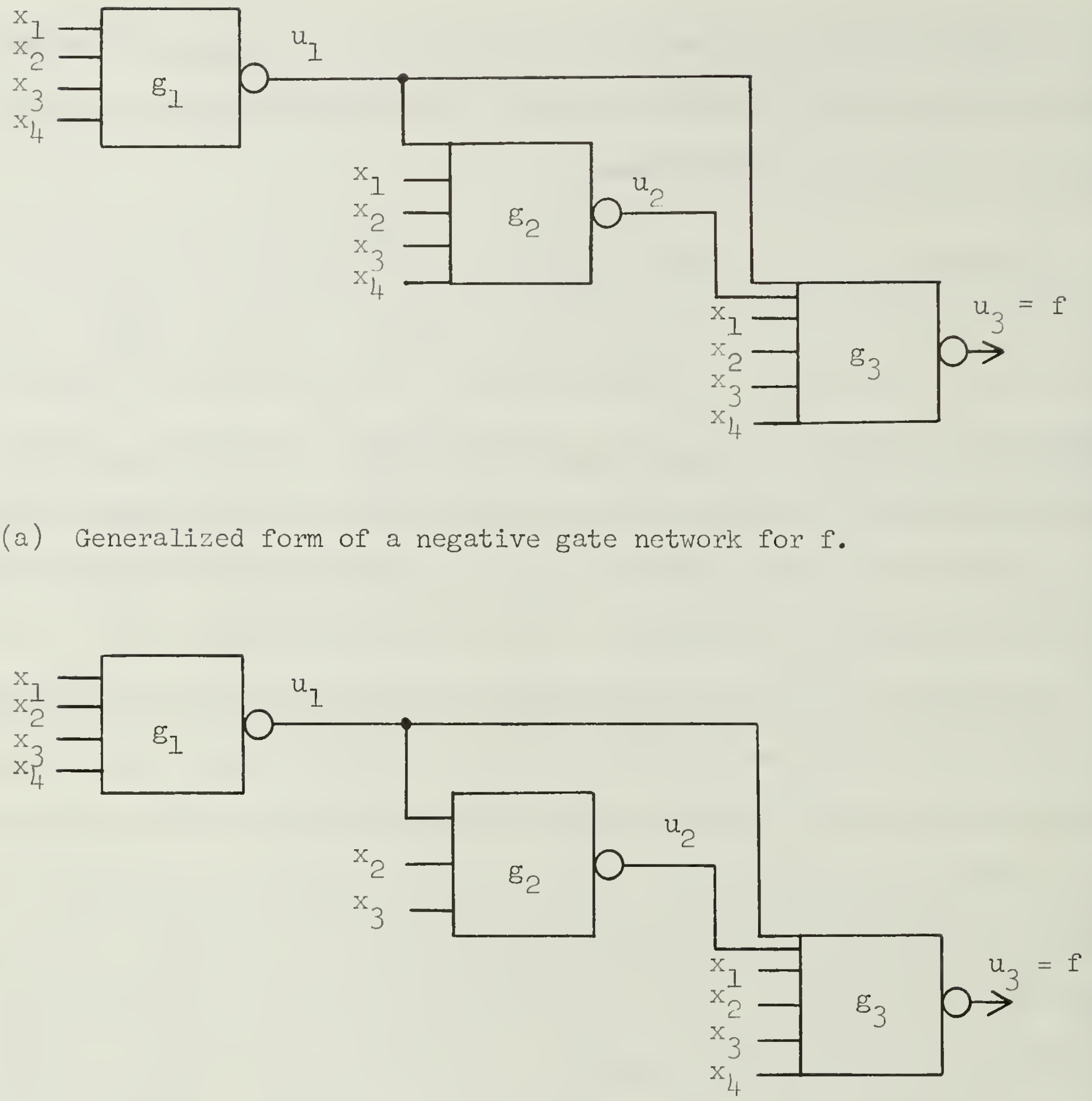
The following crucial theorem was first proved in [NTK 72]. This theorem provides the basis for multiple-level synthesis algorithms in [NTK 72], [Liu 72] (Liu uses a similar but independently derived theorem), and [Lai 76].

Theorem 4.2.1.1 A sequence of functions, $u_1,\ldots,u_R$, in C_n is an $NFS_n(R,u_R)$ if and only if the labelled n-cube with respect to $u_1,\ldots,u_R$, i.e., $C_n(u_1,\ldots,u_R)$, has no inverse edge.

Algorithm 4.1.1.2.1 (MNL) was proposed in [NTK 72] as a means of obtaining, for a given function f, a labelled n-cube, $C_n(u_1,\ldots,u_R)$ such that: (i) $u_R = f$; (ii) $C_n(u_1,\ldots,u_R)$ has no inverse edge; and (iii) the maximum bit length of the binary labels $\ell(A;u_1,\ldots,u_R)$, $A \in C_n$, is minimized, i.e., $R = R_f$. This is done by assigning the minimal possible label, $\ell(A;u_1,\ldots,u_R)$ to each vertex $A \in C_n$ under conditions (i) and (ii). It is quite obvius that Algorithm MNL actually accomplishes this goal.

An example of the n-cube labelling resulting from Algorithm MNL was shown in Fig. 4.1.1.2.2 for the function f of four variables given in Fig. 4.1.1.2.1. Note that while several inverse edges exist in the 4-cube of Fig. 4.1.1.2.1 (bold lines), none exist in Fig. 4.1.1.2.2. The $NSF_4(3,f)$ corresponds to the generalized network of three negative gates in Fig. 4.2.1.1(a). Fig. 4.2.1.1(b) shows one of the specific forms of a negative gate network corresponding to the $NFS_4(3,f)$ created by Algorithm MNL. To obtain specific forms, negative completions of the u_i with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$ must be chosen. In this particular case, u_1 and u_2 only have one negative completion each. For u_3 (as expressed in Fig. 4.2.1.1(b)) one of several possible negative completions with respect to x_1, x_2, x_3, x_4, u_1, u_2 was selected.

Basically, the G-minimal synthesis algorithm of [NTK 72] consists of applying Algorithm MNL for the given function f and obtaining negative completions of the u_i of the $NFS_n(R_f,f)$ corresponding to the resultant labelled n-cube. (Continuing the preceding example, the algorithm of [NTK 72] would



(a) Generalized form of a negative gate network for f.

(b) A negative gate network for f where:

$$\begin{aligned} u_1 &= \overline{x_1 \vee x_4\ x_2x_3} \\ u_2 &= (\bar{x}_2 \vee \bar{x}_3)(x_1 \vee x_4) = \overline{x_2x_3 \vee u_1} \\ u_3 &= x_2x_3 \vee x_1\bar{x}_2\bar{x}_4 \vee \bar{x}_1\bar{x}_3\bar{x}_4 \vee \bar{x}_1\bar{x}_2\bar{x}_3 \\ &= \overline{(u_1 \vee u_2)(x_2 \vee x_4 \vee u_1)(x_1 \vee x_3 \vee u_1)(x_1 \vee x_2 \vee x_3)}. \end{aligned}$$

Fig. 4.2.1.1 G-minimal, negative gate networks based on minimal labelling (synthesis algorithm of [NTK 72]) for function f of Fig. 4.1.1.2.2.

obtain the network of MOS cells shown in Fig. 4.2.1.2.) As will be seen, Liu's synthesis algorithm effectively produces a labelled n-cube having no inverse edges. This labelling, however, is of the special type (corresponding to the BU-stratified structure) previously mentioned which allows the easy derivation of expressions for the u_i as negative functions of their predecessors. The derivation of corresponding expressions from the more general labelled n-cubes (without inverse edges) produced in the algorithm of [NTK 72], however, requires significantly more effort.

Definition 4.2.1.2 Let $(M_0^f, M_1^f, \ldots, M_{2r}^f)$ be the BU-stratified structure of a function f. A function u is called a stratified function of f if u has the same value (0 or 1) for every input vector in the same M_i^f, $i = 0,1,\ldots,2r$. The notation $u(M_i^f)$ denotes the value of u for the input vectors in M_i^f.

It should be recalled that the clusters, M_i^f, of the BU-stratified structure are uniquely determined for a given function f.

Definition 4.2.1.3 Let $u_1,\ldots,u_R$ be a set of stratified functions of a function f. A stratified truth table with respect to f, denoted STT^f, with $u_1,\ldots,u_R$ is defined as the table in Fig. 4.2.1.3 where the entry in row M_i^f and column u_j is $u_j(M_i^f)$. If either or both of M_0^f and M_{2r}^f is empty, the corresponding row or rows may be deleted.

Examples of STT^f's are shown in Fig. 4.2.1.4.

Definition 4.2.1.4 Let $u_1,\ldots,u_R$ be stratified functions of a function f. The STT^f for $u_1,\ldots,u_R$ is called a realizable stratified truth table with respect to f, denoted $RSTT^f$, if and only if $u_1,\ldots,u_R$ is a negative function sequence of length R for f, $NFS_n(R,f)$.

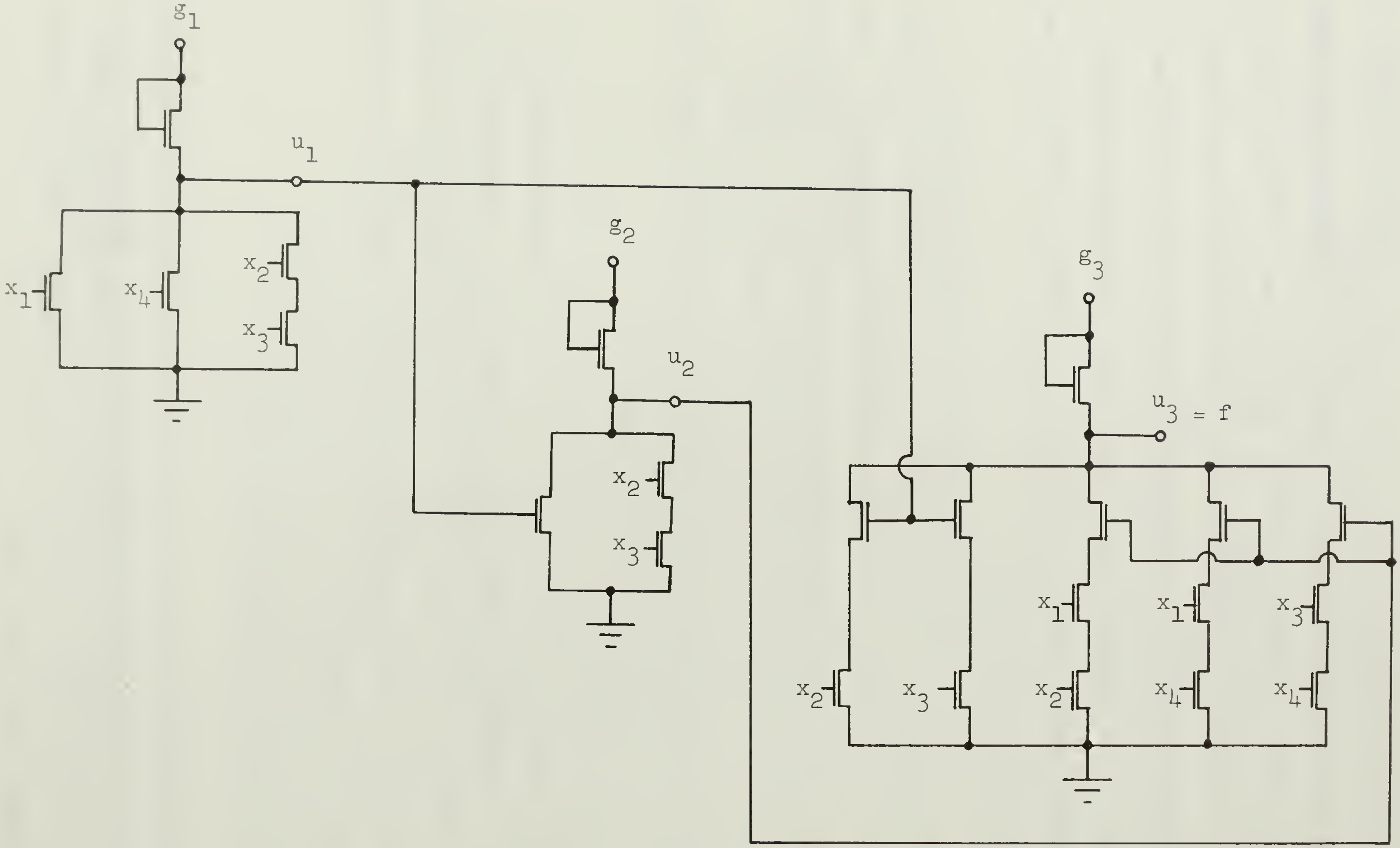


Fig. 4.2.1.2 G-minimal network of MOS cells synthesized by algorithm of [NTK 72] for function f of Fig. 4.1.1.2.1.

	u_1	u_2	$\cdots$	u_R
M^f_{2r}	$u_1(M^f_{2r})$	$u_2(M^f_{2r})$	$\cdots$	$u_R(M^f_{2r})$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
M^f_1	$u_1(M^f_1)$	$u_2(M^f_1)$	$\cdots$	$u_R(M^f_1)$
M^f_0	$u_1(M^f_0)$	$u_1(M^f_0)$	$\cdots$	$u_R(M^f_0)$

Fig. 4.2.1.3 Stratified truth table based on BU-stratified structure.

	u_1	u_2	$u_3 \not\models f$
M^f_4	0	0	1
M^f_3	0	1	0
M^f_2	0	1	1
M^f_1	1	0	1
M^f_0	1	1	0

(a) STT^f which is not an $RSTT^f$.

Fig. 4.2.1.4 Examples of stratified truth tables.

	u_1	u_2	$u_3 = f$
M_4^f	0	1	0
M_3^f	1	0	1
M_2^f	1	0	0
M_1^f	1	0	1
M_0^f	1	1	0

(b) STT^f which is not an $RSTT^f$.

	u_1	u_2	$u_3 = f$
M_4^f	0	0	0
M_3^f	0	0	1
M_2^f	0	1	0
M_1^f	1	0	1
M_0^f	1	1	0

(c) STT^f which is an $RSTT^f$.

Fig. 4.2.1.4 (Continued)

The STT^f given in Fig. 4.2.1.4(a) is not an RSTT^f since $u_R = f$. The STT^f given in Fig. 4.2.1.4(b) is not an RSTT^f since the corresponding labelled n-cube must obviously have an inverse edge (consider two vertices $A \in M_3^f$ and a $B \in M_2^f$ such that $A > B$). The STT^f given in Fig. 4.2.1.4(c) is an RSTT^f.

It is shown in [Liu 72] and is fairly obvious from previous discussions in this paper (e.g., Theorem 4.2.1.1) and others (e.g., [NTK 72]) that: an STT^f for $u_1,\ldots,u_R$ is an RSTT^f if and only if $u_R = f$ and the binary label $\ell(A;u_1,\ldots,u_R)$ is greater than $\ell(B;u_1,\ldots,u_R)$ for any pair of vertices $A \in M_i^f$, $B \in M_{i+1}^f$, $i = 0,1,\ldots,2r - 1$ (i.e., the values of the rows of the STT^f viewed as binary numbers are ascending from the top to the bottom of the table). It is also shown that an RSTT^f with only R_f columns, $u_1,\ldots,u_{R_f}$, always exists for a function f (generally, more than one exists).

<u>Definition 4.2.1.5</u> Let $u_j(M_i^f)$ be a '0' entry in an RSTT^f. Row M_k^f is said to be a <u>blocking row</u> of $u_j(M_i^f)$ if k is the largest integer such that:

(i) $k \leq i$;

(ii) $u_j(M_{k-1}^f) = 1$; and

(iii) no $\ell < j$ exists such that $u_\ell(M_i^f) > u_\ell(M_{k-1}^f)$.

If no integer k exists for which these three conditions are satisfied, blocking row of $u_j(M_i^f)$ is defined to be the bottom row of the RSTT^f.

<u>Definition 4.2.1.6</u> Let $u_j(M_i^f)$ be a '0' entry in an RSTT^f, and let row M_k^f be its blocking row. A product, $u_{j_1} \ldots u_{j_s}$, is said to be a <u>prohibiting product</u> of $u_j(M_i^f)$ if :

(i) $j_1 < \ldots < j_s < j$;

(ii) $u_{j_1}(M_i^f) = \ldots = u_{j_s}(M_i^f) = 1$;

$$\begin{array}{c|cccc}
 & u_1 & u_2 & u_3 & u_4 \\
M_7^f & 0 & 0 & 0 & 1 \\
M_6^f & 0 & 0 & 1 & 0 \\
M_5^f & 0 & 0 & 1 & 1 \\
M_4^f & 0 & 1 & 0 & 0 \\
M_3^f & \enclose{circle}{0} & 1 & \enclose{circle}{0} & 1 \\
M_3^f & 1 & 0 & 1 & 0 \\
M_1^f & 1 & 1 & 0 & 1 \\
M_0^f & 1 & 1 & 1 & \enclose{circle}{0}
\end{array}$$

$u_1(M_3^f)$. Its blocking row is M_3^f. Its prohibiting product is 1.

$u_3(M_3^f)$. Its blocking row is M_1^f. Its prohibiting product is u_2.

$u_4(M_0^f)$. Its blocking row is M_0^f. Its prohibiting products are: u_1u_3 and u_2u_3.

Fig. 4.2.1.5 An RSTT^f for a function f illustrating blocking rows and prohibiting products.

(iii) for each $\ell \geq k$ such that $u_j(M_\ell^f) = 1$, there is a $t \in \{j_1,\ldots,j_s\}$ such that $u_t(M_\ell^f) = 0$; and

(iv) if any of $u_{j_1},\ldots,u_{j_s}$ is deleted from the product, condition (iii) is no longer satisfied.

If column u_j has no '1' entry above $u_j(M_{k-1}^f)$, then the prohibiting product of $u_j(M_i^f)$ is defined to be the constant function 1.

Choosing the blocking row and a prohibiting product for a '0' entry of an $RSTT^f$ is important in Liu's generation of an expression of each u_i as a negative function of $x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$, $i = 1,\ldots,R$. Note that while a '0' entry has a unique blocking row, it may have more than one distinct prohibiting product.

Consider the '0' entry $u_1(M_3^f)$ in the $RSTT^f$ in Fig. 4.2.1.5. The blocking row for $u_1(M_3^f)$ is row M_3^f since $k = 3$ satisfies Definition 4.2.1.5. Since there is no '1' entry above $u_1(M_3^f)$, its prohibiting product is the constant function 1 by Definition 4.2.1.6. Now consider the '0' entry $u_3(M_3^f)$ in the same $RSTT^f$. Row M_3^f can not be the blocking row of $u_3(M_3^f)$ since there exists an $\ell = 2 < 3$ for which $u_\ell(M_3^f) > u_\ell(M_2^f)$. Its blocking row is row M_1^f since $u_3(M_0^f) = 1$, and no $\ell < 3$ exists such that $u_\ell(M_3^f) > u_\ell(M_0^f)$. The choice of u_2 as the prohibiting product of $u_3(M_3^f)$ can be seen to satisfy the conditions of Definition 4.2.1.6. Next, consider the '0' entry $u_4(M_0^f)$. Its blocking row must be M_0^f since no lower rows exist. Its prohibiting product may be either u_1u_3 or u_2u_3. Product $u_1u_2u_3$ violates condition (iv) of Definition 4.2.1.6, while the remaining possibilities violate condition (iii).

Based on these definitions, the synthesis algorithm of [Liu 72] for G-minimal networks can now be given.

Algorithm 4.2.1.1 Let f be a given function of n variables.

Step 1 Obtain the BU-stratified structure $(M_0^f, M_1^f, \ldots, M_{2r}^f)$. Let ζ_i^f represent the floor function of cluster M_i^f.

Step 2 Construct an RSTT^f for $u_1, \ldots, u_{R_f} = f$, $R_f = \lceil \log_2(D^f(\vec{1}, \vec{0}) + 1) \rceil + 1$ as follows:

(a) Set entry $u_{R_f}(M_i^f)$, $0 \leq i \leq 2r$, to 1 if i is an odd integer and to 0, otherwise. Omit entries and rows for M_0^f and M_{2r}^f if they are empty.

(b) Assign 0 or 1 to each remaining entry $u_j(M_i^f)$, $j \neq R_f$ such that:

$$\sum_{t=1}^{R_f} 2^{R_f - 1} u_t(M_k^f) < \sum_{t=1}^{R_f} 2^{R_f - t} u_t(M_{k-1}^f),\ k = 1, 2, \ldots, 2r$$

(i.e., such that if each row of the table is considered as a binary number, the value of each row will be greater than that of the row above).

Step 3 Obtain an expression for each u_i of $u_1, \ldots, u_{R_f}$ as a negative function of $x_1, \ldots, x_n$, $u_1, \ldots, u_{i-1}$ as follows:

(a) Set $j = 0$.

(b) Set $j = j + 1$. If $j > R_f$, go to Step 4. Otherwise, let h be the constant function 0 initially, and let I, initially, be the set of all indices i such that $u_j(M_i^f) = 0$. Go to substep (c).

(c) If $I = \emptyset$, set $\overline{u_j} = h$ and go to substep (b). Otherwise, select the smallest index i remaining in set I. Let row M_k^f be the blocking row of $u_j(M_i^f)$. Then form a new function h:

$$h \Leftarrow h \vee \zeta_k^f u_{j_1} \ldots u_{j_s},^{\dagger}$$

[†] The arrow indicates that the value of the variable on the left is to be replaced by the current value of the right-hand side.

where $u_j \ldots u_{j_s}$ is a prohibiting product of $u_j(M_i^f)$. If the prohibiting product is the constant function 1, set $\bar{u}_j = h$ and return to substep (b); otherwise, continue to substep (d).

(d) If index $\ell \in I$ exists such that

$$u_{j_1}(M_\ell^f) = \ldots = u_{j_s}(M_\ell^f) = 1,$$

set $I \Leftarrow I - \{\ell\}$. Repeat this step until no such ℓ remains in I. Return to substep (c).

Step 4 Using the irredundant disjunctive forms of the floor functions, ζ_i^f's, construct an MOS cell, g_j, from the expression of each $\bar{u}_j$, $j = 1,\ldots,R_f$, in the obvious way (i.e., series connections of FET's for conjunctions, parallel connections of FET's for disjunctions).

Example 4.2.1.1 Suppose a G-minimal network is desired for the following function of seven variables:[†]

$$\begin{aligned} f = {} & x_1\bar{x}_4 \vee x_3\bar{x}_4 \vee \bar{x}_4x_7 \vee \bar{x}_1x_3x_7 \vee x_1x_2\bar{x}_3x_4\bar{x}_5 \vee x_1x_2x_3x_4\bar{x}_7 \vee \\ & \bar{x}_1x_2x_4\bar{x}_5\bar{x}_7 \vee x_2\bar{x}_3x_4\bar{x}_5x_7 \vee \bar{x}_1x_2x_4x_5x_6x_7 \vee x_2\bar{x}_3x_4x_5x_6x_7 \vee \\ & x_1x_2x_3x_4x_5x_6x_7. \end{aligned}$$

Following Algorithm 4.2.1.1, in Step 1 it is determined that the BU-stratified structure of f consists of eight non-empty clusers, $M_0^f, M_1^f, \ldots, M_7^f$. Floor functions, expressed in irredundant disjunctive form are found to be:

[†] This function was given in an example (Example 4.4.4) in [Liu 72]. Floor functions of the clusters of the BU-stratified structure were derived, and a network was synthesized. However, the derived floor functions are not consistent with the expression given for f, probably due to typographical errors. In any case, the result given here is based on the function expression in [Liu 72] and will therefore differ from the result given in [Liu 72] (based on the floor functions) even though the same synthesis algorithm is being used.

$$\zeta_0^f = 1$$

$$\zeta_1^f = x_1 \vee x_3 \vee x_7 \vee x_2x_4$$

$$\zeta_2^f = x_1x_4 \vee x_3x_4 \vee x_4x_7 \vee x_2x_4x_5$$

$$\zeta_3^f = x_1x_2x_4 \vee x_2x_3x_4 \vee x_2x_4x_7 \vee x_3x_4x_7$$

$$\zeta_4^f = x_1x_2x_4x_5 \vee x_1x_3x_4x_7 \vee x_2x_3x_4x_5 \vee x_2x_4x_5x_7$$

$$\zeta_5^f = x_1x_2x_3x_4x_5 \vee x_2x_3x_4x_5x_7 \vee x_2x_4x_5x_6x_7$$

$$\zeta_6^f = x_1x_2x_3x_4x_5x_7$$

$$\zeta_7^f = x_1x_2x_3x_4x_5x_6x_7$$

Step 2 of the algorithm constructs an $RSTT^f$. From the BU-stratified structure, $D^f(\vec{1},\vec{0}) = 4$. Thus $R_f = \lceil \log_2(4 + 1) \rceil + 1 = 4$ columns are required in the $RSTT^f$. Fig. 4.2.1.5 shows one possible $RSTT^f$ which can be selected during Step 2 (others also exist).

Expressions for the u_i are derived in Step 3. One set of expressions consistent with the selection procedure in Step 3 is:

$$u_1 = \overline{\zeta_3^f}$$

$$u_2 = \overline{\zeta_2^f u_1 \vee \zeta_5^f}$$

$$u_3 = \overline{\zeta_1^f u_2 \vee \zeta_7^f}$$

$$u_4 = \overline{\overline{\zeta_0^f} u_1 u_3 \vee \zeta_4^f u_2 \vee \zeta_6^f u_3}$$

Step 4 of the algorithm defines the configurations of the MOS cells of the synthesized network. First, the irredundant disjunctive forms of the floor functions are substituted into the expressions of the u_i:

$$u_1 = \overline{x_1x_2x_4 \vee x_2x_3x_4 \vee x_2x_4x_7 \vee x_3x_4x_7}$$

$$u_2 = \overline{(x_1x_4 \vee x_3x_4 \vee x_4x_7 \vee x_2x_4x_5)u_1 \vee x_1x_2x_3x_4x_5 \vee}$$

$$\overline{x_2x_3x_4x_5x_7 \vee x_2x_4x_5x_6x_7}$$

$$u_3 = \overline{(x_1 \vee x_3 \vee x_7 \vee x_2x_4)u_2 \vee x_1x_2x_3x_4x_5x_6x_7}$$

$$u_4 = \overline{u_1u_3 \vee (x_1x_2x_4x_5 \vee x_1x_3x_4x_7 \vee x_2x_3x_4x_5 \vee x_2x_4x_5x_7)u_2 \vee}$$

$$\overline{(x_1x_2x_3x_4x_5x_7)u_3}.$$

From these expressions the network configuration, shown here in Fig. 4.2.1.6, can be directly obtained. The result is a G-minimal network. Factoring the expressions for the u_i can reduce the number of driver FET's in the network by approximately 37% in this case.

[Liu 72] proves that this algorithm always yields a G-minimal network for a given, completely specified function. For given, incompletely specified functions, G-minimal networks can also be obtained by ignoring unspecified vectors during the synthesis. If Algorithm 4.1.1.1.2(MXL) is used to obtain the BU-stratified structure, unspecified vectors will automatically be specified during the labelling process, and thus one need not give any special consideration to incompletely specified functions.

A similar synthesis of G-minimal networks can be carried out based on the TD-stratified structure for a given function f. Before presenting an algorithm to accomplish this synthesis, a few definitions, paralleling those for the BU-stratified structure case, must be given.

Stratified functions of f with respect to the TD-stratified structure are simply defined by substituting 'W_i^f' for every occurrence of 'M_i^f' in Definition 4.2.1.2. For the TD-stratified structure case, a stratified truth table with

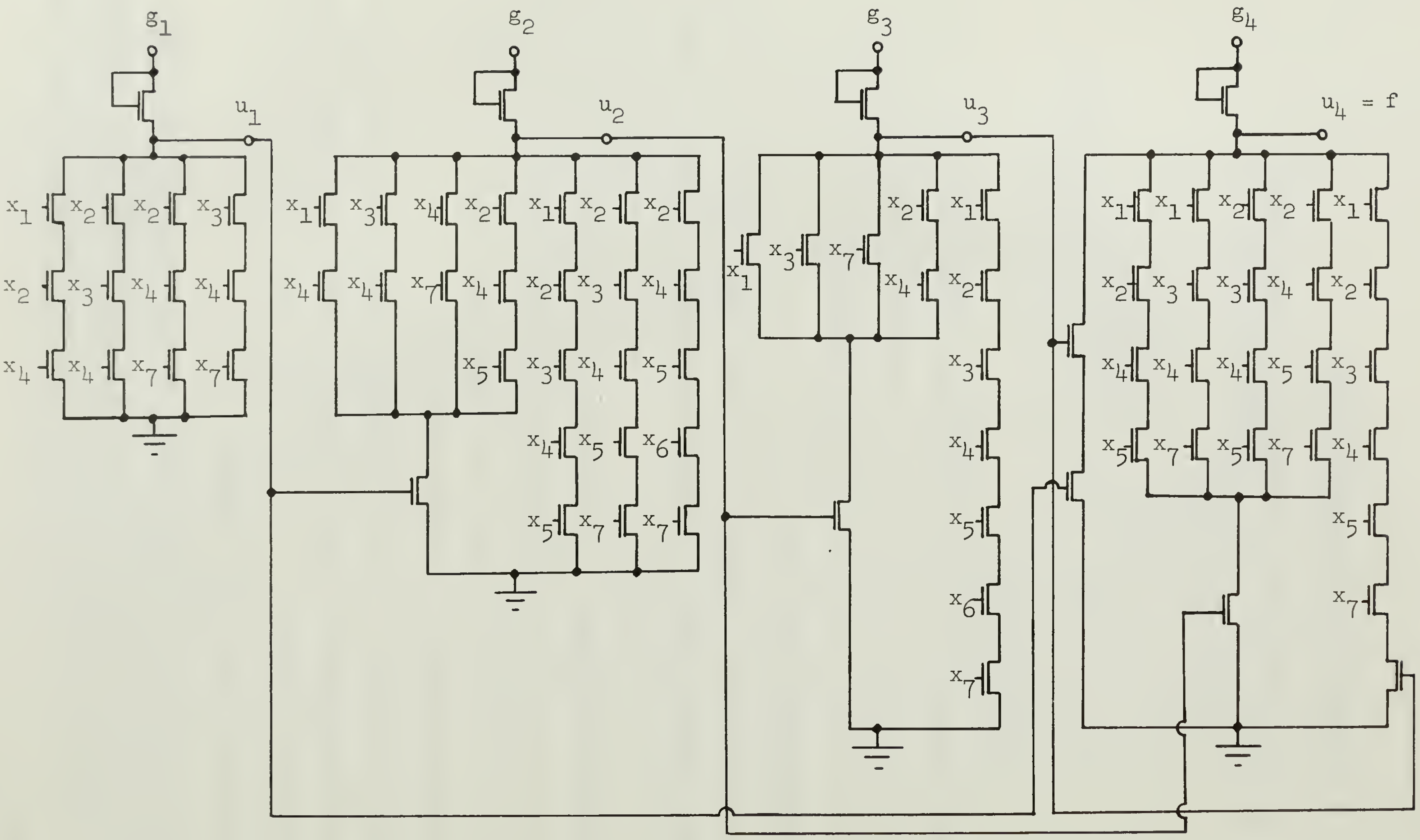


Fig. 4.2.1.6 A G-minimal network synthesized by Algorithm 4.2.1.1 for the function f in Example 4.2.1.1. Network consists of four cells and 76 driver FET's.

respect to f (corresponding to Definition 4.2.1.3) is defined as the table in Fig. 4.2.1.7. RSTT^f's (Definition 4.2.1.4) are defined for both BU- and TD-stratified structures.

Blocking rows and prohibiting products are also defined for RSTT^f's based on TD-stratified structures. In this case, however, the definitions are for '1' entries rather than for '0' entries:

Definition 4.2.1.7 Let $u_j(W_i^f)$ be a '1' entry in an RSTT^f. Row W_k^f is said to be a blocking row of $u_j(W_i^f)$ if k is the largest integer such that:

(i) $k \leq i$;

(ii) $u_j(W_{k-1}^f) = 0$; and

(iii) no $\ell < j$ exists such that $u_\ell(W_i^f) < u_\ell(W_{k-1}^f)$.

If there is no k satisfying these three conditions, the blocking row of $u_j(W_i^f)$ is defined to be the top row of the RSTT^f.

Definition 4.2.1.8 Let $u_j(W_i^f)$ be a '1' entry in an RSTT^f, and let row W_k^f be its blocking row. A product, $\bar{u}_{j_1} \ldots \bar{u}_{j_s}$ is said to be a prohibiting product of $u_j(W_i^f)$ if:

(i) $j_1 < \ldots < j_s < j$;

(ii) $u_{j_1}(W_i^f) = \ldots = u_{j_s}(W_i^f) = 0$;

(iii) for each $\ell \geq k$ such that $u_j(W_\ell^f) = 0$, there is a $t \in \{j_1,\ldots,j_s\}$ such that $u_t(W_\ell^f) = 1$; and

(iv) if any of $u_{j_1},\ldots,u_{j_s}$ is deleted from the product, condition (iii) is no longer satisfied.

If column u_j has no '0' entry below $u_j(W_{k-1}^f)$, then the prohibiting product of $u_j(W_i^f)$ is defined to be the constant function 1.

	u_1	u_2	$\cdots$	u_R
W_0^f	$u_1(W_0^f)$	$u_2(W_0^f)$	$\cdots$	$u_R(W_0^f)$
W_1^f	$u_1(W_1^f)$	$u_2(W_1^f)$	$\cdots$	$u_R(W_1^f)$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
W_{2r}^f	$u_1(W_{2r}^f)$	$u_2(W_{2r}^f)$	$\cdots$	$u_R(W_{2r}^f)$

Fig. 4.2.1.7 Stratified truth table based on TD-stratified structure.

Based on the preceding definitions, a new synthesis algorithm for G-minimal, MOS-cell networks can be given which is the counterpart of Algorithm 4.2.1.1 based on the BU-stratified structure. Since the synthesis based on TD-stratified structures so closely parallels the case of the synthesis, developed in [Liu 72], based on the BU-stratified structure, the proof of its validity will be omitted, being easily obtainable by relatively simple modifications of the proof of the validity of Algorithm 4.2.1.1 given in [Liu 72].

Algorithm 4.2.1.2 Let f be a given function of n variables.

Step 1 Obtain the TD-stratified structure $(W_0^f, W_1^f, \ldots, W_{2r}^f)$. Let θ_i^f represent the ceiling function of cluster W_i^f.

Step 2 (Same as Step 2 of Algorithm 4.2.1.1 except W_i^f is to be substituted for M_i^f.)

Step 3 Obtain an expression for each u_i of $u_1, \ldots, u_{R_f}$ as a negative function of $x_1, \ldots, x_n$, $u_1, \ldots, u_{i-1}$ as follows:

(a) Set $j = 0$.

(b) Set $j = j + 1$. If $j > R_f$, go to Step 4. Otherwise, let h be the constant function 0 initially, and let I, initially, be the set of all indices i such that $u_j(W_i^f) = 1$. Go to substep (c).

(c) If $I = \emptyset$, set $u_j = h$ and go to substep (b). Otherwise, select the smallest index i remaining in set I. Let row W_k^f be the blocking row of $u_j(W_i^f)$. Then, form a new function h:

$$h \Longleftarrow h \vee \theta_k^f \bar{u}_{j_1} \ldots \bar{u}_{j_s},$$

where $\bar{u}_{j_1} \ldots \bar{u}_{j_s}$ is a prohibiting product of $u_j(W_i^f)$. If the prohibiting product is the constant function 1, set $u_j = h$ and return to substep (b); otherwise, continue to substep (d).

(d) If index $\ell \in I$ exists such that

$$u_{j_1}(W_\ell^f) = \ldots = u_{j_s}(W_\ell^f) = 0,$$

set $I \Longleftarrow I - \{\ell\}$. Repeat this step until no such ℓ remains in I. Return to substep (c).

Step 4 (Same as Step 4 of Algorithm 4.2.1.1 except θ_i^f is to be substituted for ζ_i^f. Note, since expressions for $\bar{u}_j$ are needed here rather than the expressions for u_j derived in Step 3, De Morgan's laws should be used for quick conversion.)

For the function f of Example 4.2.1.1, both Algorithm 4.2.1.1 and Algorithm 4.2.1.2 can develop the same RSTT^f's. For the RSTT^f of Fig. 4.2.1.5, Algorithm 4.2.1.2 obtains (in Step 3) the following expressions for functions to be realized by the negative gates of the network (recall that ceiling functions can always be expressed in terms of only complemented literals):

$$u_1 = \theta_5^f$$

$$u_2 = \theta_3^f\bar{u}_1 \vee \theta_6^f$$

$$u_3 = \theta_1^f\bar{u}_2 \vee \theta_7^f$$

$$f = u_4 = \theta_0^f\bar{u}_2\bar{u}_3 \vee \theta_2^f\bar{u}_1\bar{u}_2 \vee \theta_4^f\bar{u}_3.$$

4.2.2 Synthesis methods for multiple-level networks based on stratified structures and corresponding extended floor or extended ceiling functions

This section will propose synthesis algorithms which can produce results which are improved over those obtained by the synthesis algorithms of Section 4.2.1. The improvement is a consequence of the selection of a new set of clusters, called 'extended M_i^f' ('extended W_i^f'), based on the clusters M_i^f (W_i^f)

of the BU-stratified structure (TD-stratified structure). The floor (ceiling) functions of these new clusters are called 'extended floor functions' ('extended ceiling functions'). The substitution of extended floor functions (extended ceiling functions) for floor functions (ceiling functions) in Algorithm 4.2.1.1 (4.2.1.2) gives a new algorithm which can produce improved results. In fact, the results will always be at least as good, in terms of numbers of FET's, as those derived by the algorithms of the preceding section (4.2.1). Frequently, however, the results will be significantly better.

Before giving the algorithms, the following concepts must be introduced.

Definition 4.2.2.1 Let ζ_i^f be the floor function of cluster M_i^f of the BU-stratified structure for function f of n variables. An extended floor function (EFF) of M_i^f, denoted $\zeta_i^{f'}$, is defined as any positive function of $x_1,\ldots,x_n$ for which:

(i) $\zeta_i^{f'} \supseteq \zeta_i^f$ and

(ii) for every vector A satisfying $\zeta_i^f(A) = 0$ and $\zeta_i^{f'}(A) = 1$, $f(A) = f(M_i^f)$.

A proper extended floor function (PEFF) is an extended floor function $\zeta_i^{f'}$ of M_i^f for which at least one input vector A exists satisfying $\zeta_i^f(A) = 0$ and $\zeta_i^{f'}(A) = 1$, i.e., $\zeta_i^{f'} \supset \zeta_i^f$.

Consider the function f shown with its BU-stratified structure in Fig. 4.2.2.1(a). The floor functions of $M_1^f,\ldots,M_5^f$ are:

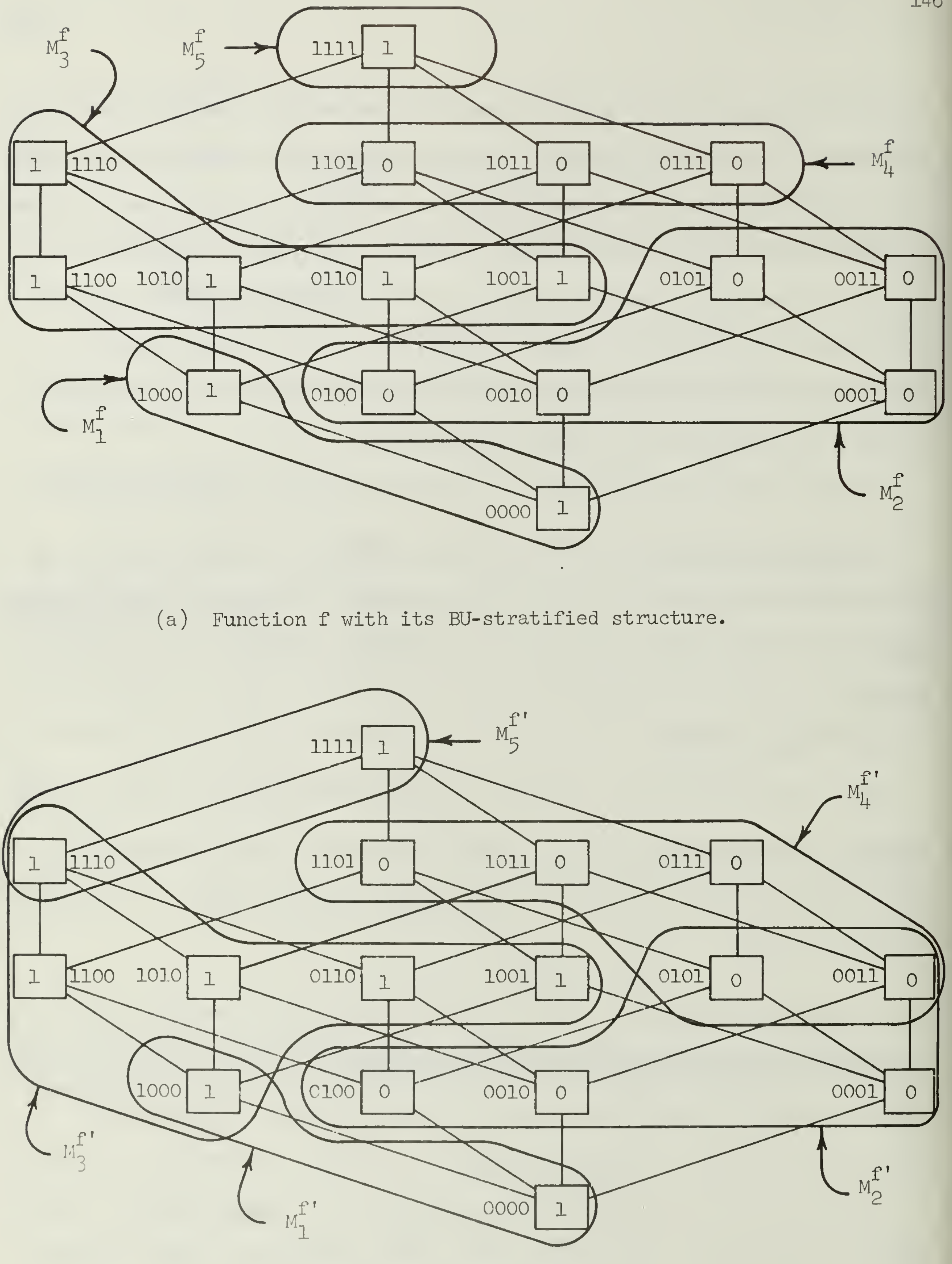
$$\zeta_1^f = 1$$

$$\zeta_2^f = x_2 \vee x_3 \vee x_4$$

$$\zeta_3^f = x_1x_2 \vee x_1x_3 \vee x_1x_4 \vee x_2x_3$$

$$\zeta_4^f = x_1x_2x_4 \vee x_1x_3x_4 \vee x_2x_3x_4$$

$$\zeta_5^f = x_1x_2x_3x_4$$



(a) Function f with its BU-stratified structure.

(b) Same function with extended M_i^f

Fig. 4.2.2.1 Example of extended M_i^f and EFF's.

The following set of EFF's can be chosen consistent with Definition 4.2.2.1:

$$\zeta_1^{f'} = 1$$

$$\zeta_2^{f'} = x_2 \vee x_3 \vee x_4$$

$$\zeta_3^{f'} = x_1 \vee x_2x_3$$

$$\zeta_4^{f'} = x_2x_4 \vee x_3x_4$$

$$\zeta_5^{f'} = x_1x_2x_3$$

In this case, $\zeta_3^{f'}$, $\zeta_4^{f'}$, and $\zeta_5^{f'}$ are all PEFF's while $\zeta_1^{f'}$ and $\zeta_2^{f'}$ are not.

The following theorem is easily seen.

Theorem 4.2.2.1 Clusters M_0^f and M_1^f of the BU-stratified structure of a function f have no proper extended floor functions. If M_0^f is empty, cluster M_2^f also has no proper extended floor function.

Consistent with this theorem, M_1^f and M_2^f in the preceding example (see Fig. 4.2.2.1(a)) were seen not to have PEFF's.

It is also fairly obvious that a cluster, M_i^f, of a BU-stratified structure has, in general, more than one possible EFF.

Definition 4.2.2.2 For a cluster M_i^f of a BU-stratified structure for a function f of n variables and a corresponding extended floor function $\zeta_i^{f'}$, an extended M_i^f (with respect to $\zeta_i^{f'}$), denoted $M_i^{f'}$, is defined as follows:

$$M_i^{f'} = M_i^f \cup \{A | A \in V_n, \zeta_i^f(A) = 0, \text{ and } \zeta_i^{f'}(A) = 1\}.$$

For the selection of EFF's in the above example, the corresponding extended M_i^f are encircled in Fig. 4.2.2.1(b).

Theorem 4.2.2.2 An extended M_i^f, $M_i^{f'}$, is always a cluster.

Proof There are two necessary conditions for a set of vectors to be a cluster (refer to Definition 4.1.1.2):

First, consider any vector $A \in M_i^{f'}$. If $A \in M_i^f$ also, $f(A) = f(M_i^f)$. If $A \notin M_i^f$, then $\zeta_i^f(A) = 0$ and $\zeta_i^{f'}(A) = 1$ from Definition 4.2.2.2. Thus, by Definition 4.2.2.1, $f(A) = f(M_i^f)$. Therefore, all vectors in $M_i^{f'}$ are of the same type (i.e., all true or all false vectors), satisfying the first necessary condition for $M_i^{f'}$ to be a cluster.

The remaining condition for $M_i^{f'}$ to be a cluster is that for every pair of vectors $A, B \in M_i^{f'}$, no vector $C \notin M_i^{f'}$ exists such that:

$$A > C > B.$$

Suppose such a C does exist for some pair A,B. It will be shown that this supposition always leads to a contradiction. Since $C \notin M_i^{f'}$, $C \notin M_i^f$ also. By the definition of $M_i^{f'}$ and the fact that $\zeta_i^{f'} \supseteq \zeta_i^f$, $\zeta_i^{f'}(D) = 1$ for every vector D in $M_i^{f'}$. Hence, $\zeta_i^{f'}(B) = 1$, and, consequently, $\zeta_i^{f'}(C) = 1$ since $\zeta_i^{f'}$ is a positive function and $C > B$. If $\zeta_i^f(C) = 0$ also, C would be a member of $M_i^{f'}$ by Definition 4.2.2.2 (a contradiction). So assume $\zeta_i^f(C) = 1$. Since ζ_i^f can be expressed as the disjunction of β-terms corresponding to the floor vectors of M_i^f (by Definition 4.1.1.4), there must exist a floor vector E of M_i^f such that $E < C$ ($E = C$ can not occur since $C \notin M_i^f$). However, $E < C < A$ where $E, A \in M_i^f$ and $C \notin M_i^f$ is a contradiction of the fact that M_i^f is a cluster. Therefore, no such $C \notin M_i^f$ can exist, and the theorem statement is correct.

Q.E.D.

Definition 4.2.2.3 The extended range of an EFF, $\zeta_i^{f'}$, of a cluster M_i^f of the BU-stratified structure for a function f is defined to be cluster M_j^f where j is the smallest index such that there exists a vector $A \in M_j^f$ satisfying

$\zeta_i^{f'}(A) = 1$. A vector B and the cluster, M_k^f, to which it belongs are said to be within the extended range of an EFF, $\zeta_i^{f'}$, if and only if $\zeta_i^{f'}(B) = 1$ and $k < i$.

Consider again the M_i^f and EFF's of the example illustrated in Fig. 4.2.2.1. M_3^f is in the extended range of $\zeta_5^{f'}$. Vector (0011) is in the extended range of $\zeta_4^{f'}$, and cluster M_1^f is the extended range of $\zeta_3^{f'}$. Although not seen in this example, it is possible that clusters M_j^f and M_k^f can be in the extended range of some $\zeta_i^{f'}$ while a M_ℓ^f, $j > \ell > k$, is not.

Theorem 4.2.2.3 If M_i^f and M_j^f are any two clusters of the BU-stratified structure for a function f such that M_j^f is in the extended range of an EFF, $\zeta_i^{f'}$, of M_i^f, then i - j is an even integer.

Proof Consider any vector $A \in M_j^f$ which is in the extended range of $\zeta_i^{f'}$. $\zeta_i^f(A) = 0$, since $\zeta_i^f(A) = 1$ would imply that $A \geq B$ for some floor vector B of M_i^f, violating Property II of the BU-stratified structure. By definition of being in the extended range of an EFF, $\zeta_i^{f'}(A) = 1$. Thus, $A \in M_i^{f'}$, the extended M_i^f. Since $M_i^f \subseteq M_i^{f'}$ and $M_i^{f'}$ is a cluster, $f(A) = f(M_i^f)$. Considering $f(A) = f(M_j^f)$ also, M_i^f and M_j^f must both be true clusters or both be false clusters of f, i.e., i - j must be an even integer.

Q.E.D.

As seen by the above example, expressions for EFF's may have significantly fewer literals than expressions for the corresponding floor functions (it is also possible that they can have more, but such EFF's will not be selected by the methods given in this section). Since in Algorithm 4.2.1.1 each literal corresponds to an FET of the synthesized network, the ability to substitute

EFF's for floor functions in the algorithm can lead to substantial reductions in the numbers of FET's in the synthesized networks (later examples will show reductions of as much as 29%).

In general, EFF's can not be freely substituted for floor functions in the expressions for the u_i developed in Algorithm 4.2.1.1 without causing the synthesized network to realize some function other than the intended one. Conditions will be given under which a particular EFF can be substituted for the corresponding floor function. Actually, however, such a substitution can be made under much wider circumstances which are significantly more complex to state and to check. Since empirical evidence seems to suggest that situations in which EFF's can not be used exclusively in place of floor functions are infrequent, it may be more practical to actually test whether the substitution results in as incorrect network output function rather than to attempt to check whether one of several complex conditions exists. A simple method to accomplish this test will be given later and demonstrated.

Theorem 4.2.2.4 Consider an expression for a function u_j developed by Step 3 of Algorithm 4.2.1.1 during the synthesis of a network for a given function f. An EFF $\zeta_i^{f'}$ with extended range M_{i-2q}^f can be substituted for floor function ζ_i^f, $0 \leq i \leq 2r$, note that not all $\zeta_i^{f'}$ need be substituted in this expression if, for each t = 1,2,...,q, either:

(1) M_{i-2t}^f is not in the extended range of $\zeta_i^{f'}$;

(2) $u_j(M_{i-2t}^f) = 0$; or

(3) if the prohibiting product for row M_i^f is $u_{j_1} \ldots u_{j_s}$, $u_{j_k}(M_{i-2t}^f) = 0$ for at least one k, $i \leq k \leq s$.

Proof The expression developed by Algorithm 4.2.1.1 for function u_j can be written as:

$$\overline{Y \vee u_{j_1} \ldots u_{j_s} \zeta_i^f}, \tag{4.1}$$

where there is no occurrence of ζ_i^f in the expression Y (it is clear from Steps 3c and 3d of the algorithm that no ζ_i^f can appear twice in the expression for u_j).

ζ_i^f and $\zeta_i^{f'}$ can both be represented by disjunctions of minterms corresponding to their respective true vectors. Since $\zeta_i^{f'} \supseteq \zeta_i^f$, every minterm of ζ_i^f is a minterm of $\zeta_i^{f'}$. So replacing ζ_i^f by $\zeta_i^{f'}$ in expression (4.1) for u_j is equivalent to writing:

$$\overline{Y \vee u_{j_1} \ldots u_{j_s} \zeta_i^f \vee u_{j_1} \ldots u_{j_s} \gamma_1 \vee \ldots \vee u_{j_1} \ldots u_{j_s} \gamma_p} \tag{4.2}$$

where γ_ℓ represents the minterm corresponding to vector A_ℓ for which $\zeta_i^f(A_\ell) = 0$ and $\zeta_i^{f'}(A_\ell) = 1$. It will be shown that (4.2) is an alternate expression for u_j under the conditions of the theorem.

Obviously, the equivalence of expressions (4.1) and (4.2) can only be in doubt for vectors of the set $\{A_1, \ldots, A_p\}$, i.e., vectors in the extended range of $\zeta_i^{f'}$, i.e., certain vectors in clusters $M_{i-2}^f, M_{i-4}^f, \ldots, M_{i-2q}^f$. Thus, it is sufficient to show that (4.1) and (4.2) have identical values for each vector in M_{i-2t}^f, $t = 1,2,\ldots,q$, when at least one of the three conditions of the theorem statement are met.

Expressions (4.1) and (4.2) clearly have identical values for each vector in M_{i-2t}^f, $1 \leq t < q$, if M_{i-2t}^f is not in the extended range of $\zeta_i^{f'}$ (condition (1) of the theorem).

Now consider each vector A_ℓ, $\ell = 1,\ldots,p$, in the extended range of $\zeta_i^{f'}$. Let $A_\ell \in M_{i-2t}^f$ and assume either condition (2) or (3) is met by M_{i-2t}^f. If condition (2) is true for M_{i-2t}^f, then $u_j(A_\ell) = 0$. Hence, $Y \vee u_{j_1} \ldots u_{j_s} \zeta_i^f = 1$ for vector A_ℓ (by expression (4.1)), and expression (4.2) also has the value

0 regardless of the values of its other terms (e.g., $u_{j_1} \ldots u_{j_s} \gamma_\ell$). If condition (3) is true for M^f_{i-2t}, $u_{j_k}(A_\ell) = 0$ must hold for at least one k, $1 \leq k \leq s$. Thus, also under this condition, expressions (4.1) and (4.2) clearly have identical values for every vector in V_n.

Therefore, if at least one on the three conditions of the theorem is met by every M^f_{i-2t}, $t = 1,2,\ldots,q$, ζ^f_i can always be replaced by $\zeta^{f'}_i$ in an expression for u_j as developed by Algorithm 4.2.1.1. Clearly the same argument can be made for substitutions of $\zeta^{f'}_i$ for ζ^f_i for additional i.

Q.E.D.

Corollary 4.2.2.1 An extended floor function $\zeta^{f'}_i$ can be substituted for each corresponding floor function ζ^f_i appearing in the expression for u_{R_f} (i.e., the function to be realized by the output gate or cell) developed by Step 3 of Algorithm 4.2.1.1 during the synthesis of a G-minimal network for a given function f.

Proof Since only ζ^f_i corresponding to M^f_i for which $u_{R_f}(M^f_i) = f(M^f_i) = 0$ can appear in the expression for u_{R_f}, every cluster in the extended range of $\zeta^{f'}_i$ must be a false cluster for f. Therefore, condition (1) or (2) of Theorem 4.2.2.4 is satisfied for each cluster M^f_{i-2t}, $t = 1,2,\ldots,q$, where M^f_{i-2q} is the extended range of $\zeta^{f'}_i$.

Q.E.D.

Theorem 4.2.2.4 permits the use of EFF's under conditons which maintain the same functions u_j, $j = 1,\ldots,R_f$, constructed by Algorithm 4.2.1.1 for a given function f. Actually, however, functions u_j for $j = 1,\ldots,R_f - 1$ may be changed as long as u_{R_f} remains equal to f. This may seem difficult since u_{R_f} is dependent on $u_1,\ldots,u_{R_f-1}$ (as well as $x_1,\ldots,x_n$), but it has been found (by empirical evidence) that the changes in the functions u_j, $j = 1,\ldots,R_f - 1$, caused by substituting EFF's for floor functions in the expressions (as derived by Algorithm 4.2.1.1) for the u_j often do not induce changes in u_{R_f} (i.e., do not cause the synthesized network to realize an incorrect function). Hence, Theorem 4.2.2.4 and Corollary 4.2.2.1 cover only a portion of the cases in which EFF's can be used.

Example 4.2.2.1 Consider the synthesis described in Example 4.2.1.1 for a function f of four variables. The RSTT^f selected in Step 2 of Algorithm 4.2.1.1 is shown in Fig. 4.2.1.5, and the expressions for $u_1,\ldots,u_4$ chosen in Step 3 are:

$$u_1 = \bar{\zeta}_3^f$$

$$u_2 = \overline{\zeta_2^f u_1 \vee \zeta_5^f}$$

$$u_3 = \overline{\zeta_1^f u_2 \vee \zeta_7^f}$$

$$u_4 = \overline{\zeta_0^f u_1 u_3 \vee \zeta_4^f u_2 \vee \zeta_6^f u_3}$$

Suppose the EFF's shown in Table 4.2.2.1 are chosen for the clusters, M_i^f, of the BU-stratified structure of f. The difference between the number of literals in the expression for an EFF and that in the expression for the corresponding floor function (see Table 4.2.2.1) is exactly the savings, in terms of numbers of FET's, which can be achieved by each replacement of an occurrence of the floor function by the corresponding EFF in the expressions for u_j determined in Step 3 of the algorithm. In this case, only ζ_1^f for u_3 and ζ_0^f, ζ_4^f, ζ_6^f for u_4 are found to satisfy the conditions of Theorem 4.2.2.4 for replacement by their respective EFF's. However, since only $\zeta_4^{f'}$ and $\zeta_6^{f'}$ of Table 4.2.2.1 are PEFF's only they can contribute to a reduction of the number of FET's in the synthesized network: 11 FET, in this case.

Choosing alternate RSTT^f's can sometimes improve the number of floor

$\zeta_i^{f'}$	clusters within extended range of $\zeta_i^{f'}$	literals in expression for ζ_i^{f}	literals in expression for $\zeta_i^{f'}$	difference in numbers of literals
$\zeta_7^{f'} = x_2x_5x_6x_7$	M_1^f, M_5^f	7	4	3
$\zeta_6^{f'} = x_1x_3x_4x_7$	M_4^f	6	4	2
$\zeta_5^{f'} = x_1x_2x_3x_5 \vee x_2x_3x_5x_7 \vee x_2x_5x_6x_7$	M_1^f	15	12	3
$\zeta_4^{f'} = x_2x_4x_5 \vee x_1x_3x_4x_7$	M_2^f	16	7	9
$\zeta_3^{f'} = x_1x_2 \vee x_2x_3 \vee x_2x_7 \vee x_3x_7$	M_1^f	12	8	4
$\zeta_2^{f'} = x_1x_4 \vee x_3x_4 \vee x_4x_5 \vee x_4x_7$	M_0^f	9	8	1
$\zeta_1^{f'} = x_1 \vee x_3 \vee x_7 \vee x_2x_4$	—	5	5	0
$\zeta_0^{f'} = 1$	—	0	0	0

Table 4.2.2.1 Extended floor functions for function f of Examples 4.2.1.1 and 4.2.2.1.

functions which are replacable by EFF's. For the same function f the RSTT^f shown in Fig. 4.2.2.2 may also be chosen in Step 2 of the algorithm. Corresponding expressions for the u_j are subsequently developed in Step 3:

$$u_1 = \overline{\zeta_3^f}$$
$$u_2 = \overline{\zeta_1^f u_1 \vee \zeta_5^f}$$
$$u_3 = \overline{\zeta_7^f}$$
$$u_4 = \overline{\zeta_0^f u_1 u_2 \vee \zeta_2^f u_1 \vee \zeta_4^f u_2 \vee \zeta_6^f u_3}$$

For this combination of RSTT^f and u_j, all EFF's except $\zeta_3^{f'}$ and $\zeta_7^{f'}$ may be used under the conditions of Theorem 4.2.2.4. This enables the synthesis of a network with 15 fewer driver FET's (than that synthesized without the use of EFF's) – a reduction of 20%.

In the preceding example, expressions for EFF's all contained numbers of literals smaller than or equal to those of their floor function counterparts. In general, for an arbitrary choice of EFF's, this need not be true. The following algorithm, however, insures the selection of an EFF with the same or smaller number of literals than the corresponding floor function of a particular cluster.

Algorithm 4.2.2.1 Algorithm for the selection of extended floor functions (SEFF). Let M_i^f of the BU-stratified structure for a function f of n variables be the cluster for which an EFF, $\zeta_i^{f'}$, is desired.

Step 1 Initially, let: E be the set of floor vectors of cluster M_i^f; H be the set of vectors in M_i^f; and I be the null set.

	u_1	u_2	u_3	$u_4 = f$
M_7^f	0	0	0	1
M_6^f	0	0	1	0
M_5^f	0	0	1	1
M_4^f	0	1	1	0
M_3^f	0	1	1	1
M_2^f	1	0	1	0
M_1^f	1	0	1	1
M_0^f	1	1	1	0

Fig. 4.2.2.2 An alternate $RSTT^f$ for function f of Examples 4.2.1.1 and 4.2.2.1.

Step 2 Select a vector A from set E. Seek a vector B of minimal weight such that $B \leq A$ holds and $H \cup \{C | C \in V_n, B \leq C \leq D$ for at least one vector $D \in H\}$ is a cluster.

Step 3 $E \Leftarrow E - \{D | D \in E$ and $D \geq B\}$. $H \Leftarrow H \cup \{C | C \in V_n, B \leq C \leq D$ for at least one vector $D \in H\}$. $I \Leftarrow I \cup \{B\}$.

Step 4 If set E is not empty, return to Step 2. Otherwise, set:

$$\zeta_i^{f'} = \beta_1 \vee \cdots \vee \beta_p$$

where β_i is the β-term of vector B_i in the set I and $|I| = p$. Also, $M_i^{f'} = H$.

Since each floor vector A of M_i^f whose β-term appears in the disjunction expressing ζ_i^f is replaced by a vector B (sometimes several floor vectors of M_i^f are replaced by a single vector B) of at most the same weight (because $B \leq A$), the expression for $\zeta_i^{f'}$ will always have a number of literals less than or equal to that of ζ_i^f. It is clear from the algorithm that there exist no vectors in $M_i^{f'} - I$ which are less than vectors in I and that every vector in $M_i^{f'} - I$ is greater than at least one vector in I. To show that I is the set of all floor vectors of $M_i^{f'}$, it only remains to show that no pair of vectors, $B_1, B_2 \in I$ exists such that $B_1 < B_2$. This cannot occur, since B_2 would not be selected in Step 2 if such a B_1 existed.

As previously mentioned, a complete substitution of EFF's for floor functions is often possible even when the conditions of Theorem 4.2.2.4 are not satisfied. The following algorithm is given to simply test the feasibility of such a substitution for a given function f, $RSTT^f$, and set of EFF's.

Algorithm 4.2.2.2 Algorithm to test the feasibility of the replacement of floor functions by EFF's in the expressions for u_j developed in Algorithm 4.2.1.1 (TFEFF).

Let $(M_0^f, M_1^f, \ldots, M_{2r}^f)$ be the BU-stratified structure for a given function f and let $\zeta_0^{f'}, \zeta_1^{f'}, \ldots, \zeta_{2r}^{f'}$ be the set of EFF's proposed for substitution, in place of occurrences of $\zeta_0^f, \zeta_1^f, \ldots, \zeta_{2r}^f$, respectively, into the expression u_j, $j = 1, \ldots, R_f$, developed in Step 3 of Algorithm 4.2.1.1.

Step 1 Prepare a blank truth table of the following type (see Fig. 4.2.2.3):

(a) For each cluster M_i^f, $i = 0, 1, \ldots, 2r$, determined to be within the extended range of EFF's $\zeta_{k_1}^{f'}, \ldots, \zeta_{k_{p_i}}^{f'}$, create a group of one or more adjacent rows of the truth table (call this group rows of M_i^f one row corresponding to each non-empty set among the 2^{p_i} sets: $M_i^f \cap M_{k_1}^{f'} \cap \ldots \cap M_{k_{p_i-1}}^{f'} \cap M_{k_{p_i}}^{f'}$; $M_i^f \cap M_{k_1}^{f'} \cap \ldots \cap M_{k_{p_i-1}}^{f'} \cap \bar{M}_{k_{p_i}}^{f'}$; $M_i^f \cap M_{k_1}^{f'} \cap \ldots \cap \bar{M}_{k_{p_i-1}}^{f'} \cap M_{k_{p_i}}^{f'}$; $M_i^f \cap M_{k_1}^{f'} \cap \ldots \cap \bar{M}_{k_{p_i-1}}^{f'} \cap \bar{M}_{k_{p_i}}^{f'}$; ... ; $M_i^f \cap \bar{M}_{k_1}^{f'} \cap \ldots \cap \bar{M}_{k_{p_i-1}}^{f'} \cap M_{k_{p_i}}^{f'}$; $M_i^f \cap \bar{M}_{k_1}^{f'} \cap \ldots \cap \bar{M}_{k_{p_i-1}}^{f'} \cap \bar{M}_{k_{p_i}}^{f'}$; where $\bar{M}_k^f$ denoted the set $V_n - M_k^f$.

(b) For each function u_j, $j = 1, \ldots, R_f$, for which an expression has been determined in Step 3 of Algorithm 4.2.1.1, create a column of the truth table. Let e_{ij} denote the entry at the intersection of row i and column u_j.

Step 2 Determine the entries of the truth table.

(a) $j = 0$.

(b) $j = j + 1$. If $j > R_f$, go to Step 3.

(c) For expression $\bar{u}_j$, determine the 0 entries of column u_j as follows. Consider each term $u_{\ell_1} \ldots u_{\ell_{q_i}} \zeta_i^f$ in the expression: For each row k among (i) all rows of M_i^f and above and (ii) those rows of M_{i-1}^f and below representing vectors in the extended range of ζ_i^f (i.e., rows corresponding to non-empty sets in the intersection of $M_i^{f'}$ with other clusters and extended clusters), set $e_{kj} = 0$ if and only if $e_{k,\ell_1} = \ldots = e_{k,\ell_{q_i}} = 1$.

(d) For all remaining blank entries in column u_j, set $e_{kj} = 1$.

Step 3 EFF's may completely replace floor functions in the expressions developed in Step 3 of Algorithm 4.2.1.1 without affecting the output function of the synthesized network if and only if, in column u_{R_f}, all entries for rows of M_{2i}^f, $i = 0,1,\ldots,r$, are 0 and all entries for rows of M_{2i-1}^f, $i = 1,2,\ldots,r$, are 1.

It is simple to see the validity of this algorithm when it is realized that the calculated truth table is just a 'condensed' truth table for the functions realized by the gates (cells) of the synthesized network resulting from Algorithm 4.2.1.1 after substituting EFF's for floor functions. If the condition in Step 3 of Algorithm TFEFF is not met, it implies that the network will have the incorrect output 0 for a true vector of function f or the incorrect output 1 for a false vector of f.

Example 4.2.2.2 Reconsider the synthesis of a G-minimal network to realize the seven-variable function given in Example 4.2.1.1. Let the set of EFF's , $\zeta_0^{f'},\ldots,\zeta_7^{f'}$, in Table 4.2.2.1 be proposed replacements for the floor functions, $\zeta_0^f,\ldots,\zeta_7^f$, in the expressions for $\bar{u}_1,\ldots,\bar{u}_4$ developed in Example 4.2.1.1:

$$\bar{u}_1 = \zeta_3^f$$

$$\bar{u}_2 = \zeta_2^f u_1 \vee \zeta_5^f$$

$$\bar{u}_3 = \zeta_1^f u_2 \vee \zeta_7^f$$

$$\bar{u}_4 = \zeta_0^f u_1 u_3 \vee \zeta_4^f u_2 \vee \zeta_6^f u_3.$$

		u_1	u_2	$\dots$	u_{R_f}
rows of M^f_{2r}	M^f_{2r}				
rows of M^f_{2r-1}	M^f_{2r-1}				
rows of M^f_{2r-2}	$M^f_{2r-2} \cap M^{f'}_{2r}$				
	$M^f_{2r-2} \cap \bar{M}^{f'}_{2r}$				
rows of M^f_{2r-3}	$M^f_{2r-3} \cap M^{f'}_{2r-1}$				
	$M^f_{2r-3} \cap \bar{M}^{f'}_{2r-1}$				
rows of M^f_{2r-4}	$M^f_{2r-4} \cap M^{f'}_{2r} \cap M^{f'}_{2r-2}$				
	$M^f_{2r-4} \cap M^{f'}_{2r} \cap \bar{M}^{f'}_{2r-2}$				
	$M^f_{2r-4} \cap \bar{M}^{f'}_{2r} \cap M^{f'}_{2r-2}$				
	$M^f_{2r-4} \cap \bar{M}^{f'}_{2r} \cap \bar{M}^{f'}_{2r-2}$				
rows of M^f_0	$\vdots$			$\vdots$	
	$M^f_0 \cap M^{f'}_{2r} \cap \dots \cap M^{f'}_2$				
	$\vdots$				
	$M^f_0 \cap \bar{M}^{f'}_{2r} \cap \dots \cap \bar{M}^{f'}_2$				

Fig. 4.2.2.3 Generalized truth table constructed by Algorithm 4.2.2.2 (TFEFF). Rows corresponding to empty sets will be omitted in the actual truth table.

Use Algorithm 4.2.2.2 (TFEFF) to test the feasibility of the proposed substitution. The truth table shown in Fig. 4.2.2.4 is constructed in accordance with the algorithm.

In Step 1, it is found that: vectors (0101111), (0111111), (1101111) of M_5^f are in $M_7^{f'}$; vectors (1011001), (1011011), (1011101), (1011111), (1111001), (1111011) of M_4^f are in $M_6^{f'}$; vectors (0101100), (0101110) of M_2^f are in $M_4^{f'}$; vectors (0100111), (0110111), (1100111), (1110111) of M_1^f are in $M_3^{f'} \cap M_5^{f'} \cap M_7^{f'}$; vectors (1110100), (1110101), (1110110) of M_1^f are in $M_3^{f'} \cap M_5^{f'} \cap \overline{M}_7^{f'}$; vector (0010011) and many others of M_1^f are in $M_3^{f'} \cap \overline{M}_5^{f'} \cap \overline{M}_7^{f'}$. Also each cluster has at least one vector not in the extended range of an EFF of another cluster. This determines the 15 rows in 8 groups shown in the figure. There is one column for each of the four functions $u_1,\ldots,u_4$.

Step 2 determines the entries of the truth table as shown in the figure. Finally, the condition in Step 3 of the algorithm is seen to be satisfied by the truth table, allowing the complete replacement of the floor functions by the EFF's:

$$\begin{aligned}
\bar{u}_1 &= \zeta_3^{f'} \\
\bar{u}_2 &= \zeta_2^{f'} u_1 \vee \zeta_5^{f'} \\
\bar{u}_3 &= \zeta_1^{f'} u_2 \vee \zeta_7^{f'} \\
\bar{u}_4 &= \zeta_0^{f'} u_1 u_3 \vee \zeta_4^{f'} u_2 \vee \zeta_6^{f'} u_3 .
\end{aligned}$$

Continuing the synthesis of a G-minimal network with these expressions, Step 4 of Algorithm 4.2.1.1 obtains:

$$u_1 = \overline{x_1x_2 \vee x_2x_3 \vee x_2x_7 \vee x_3x_7}$$

$$u_2 = \overline{(x_1x_4 \vee x_3x_4 \vee x_4x_5 \vee x_4x_7)u_1 \vee x_1x_2x_3x_5 \vee x_2x_3x_5x_7 \vee x_2x_5x_6x_7}$$

$$u_3 = \overline{(x_1 \vee x_3 \vee x_7 \vee x_2x_4)u_2 \vee x_2x_5x_6x_7}$$

$$u_4 = \overline{u_1u_3 \vee (x_2x_4x_5 \vee x_1x_3x_4x_7)u_2 \vee x_1x_3x_4x_7u_3}$$

From these expressions, the network configuration shown in Fig. 4.2.2.5 is obtained. The synthesized network consists of four cells and 54 driver FET's. This represents a 29% reduction in the number of driver FET's compared to the previous result obtained without the use of EFF's (see Fig. 4.2.1.6). Of course, the expressions defining the network configurations can be factored to improve both results, but there will still be 17% fewer driver FET's required for the design using EFF's (an additional consideration is that expressions containing the simpler EFF's are easier to factor).

Concepts were developed in this section for the improvement of the results obtained by Algorithm 4.2.1.1 which is based on BU-stratified structures. Similarly, concepts of extended ceiling functions, extended W_i^f's, etc., can be developed for TD-stratified structures and thus enable a corresponding improvement of the results obtained by Algorithm 4.2.1.2.

	u_1	u_2	u_3	u_4
M_7^f	0	0	0	1
M_6^f	0	0	1	0
$M_5^f \cap M_7^{f'}$	0	0	0	1
$M_5^f \cap \overline{M}_7^{f'}$	0	0	1	1
$M_4^f \cap M_6^{f'}$	0	1	0	0
$M_4^f \cap \overline{M}_6^{f'}$	0	1	0	0
M_3^f	0	1	0	1
$M_2^f \cap M_4^{f'}$	1	0	1	0
$M_2^f \cap \overline{M}_4^{f'}$	1	0	1	0
$M_1^f \cap M_3^{f'} \cap M_5^{f'} \cap M_7^{f'}$	0	0	0	1
$M_1^f \cap M_3^{f'} \cap M_5^{f'} \cap \overline{M}_7^{f'}$	0	0	1	1
$M_1^f \cap M_3^{f'} \cap \overline{M}_5^{f'} \cap \overline{M}_7^{f'}$	0	1	0	1
$M_1^f \cap \overline{M}_3^{f'} \cap \overline{M}_5^{f'} \cap \overline{M}_7^{f'}$	1	1	0	1
$M_0^f \cap M_2^{f'}$	1	0	1	0
$M_0^f \cap \overline{M}_2^{f'}$	1	1	1	0

Fig. 4.2.2.4 Truth table constructed by Algorithm 4.2.2.2 (TFEFF) for Example 4.2.2.2.

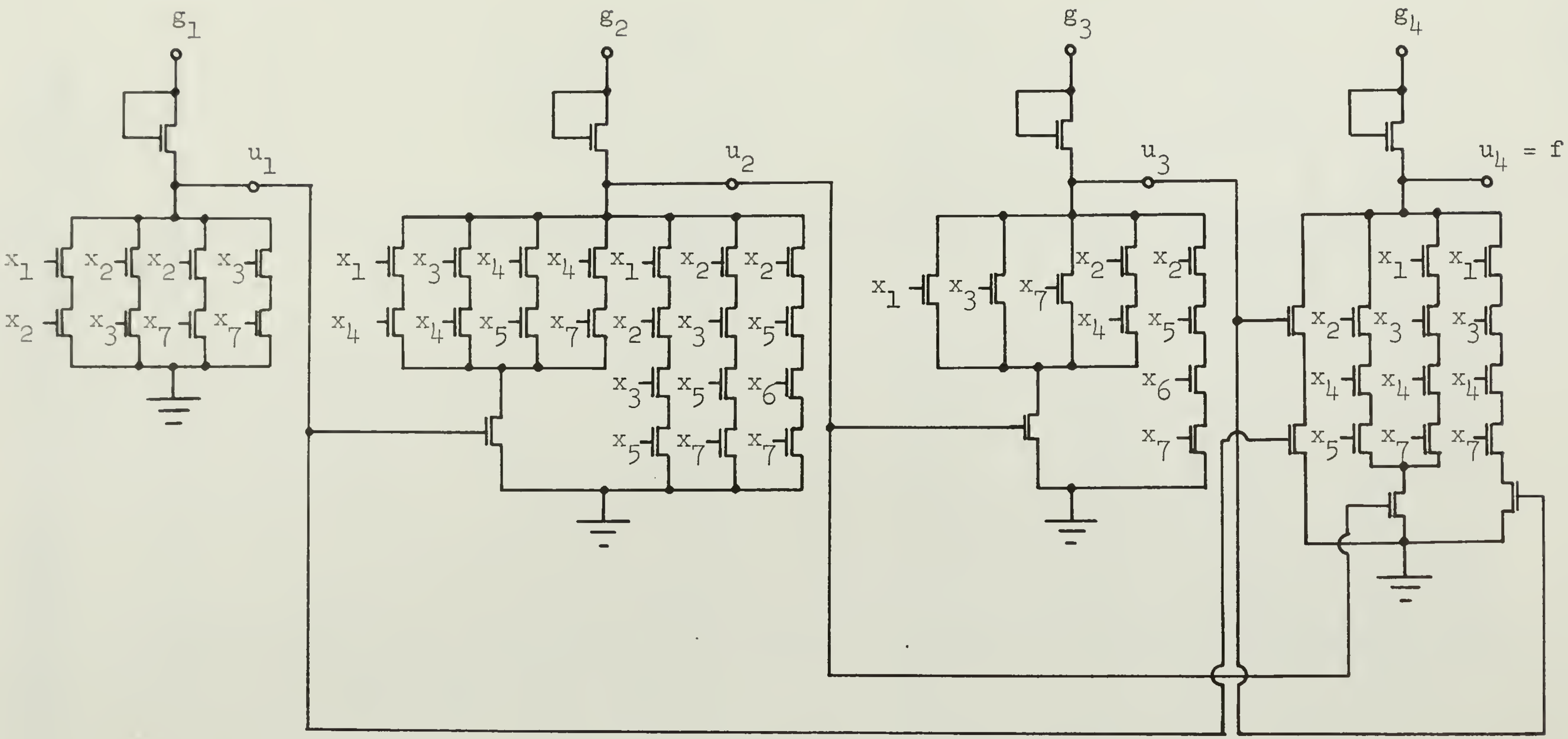


Fig. 4.2.2.5 G-minimal network obtained for function f of Example 4.2.1.1 by the combination of Algorithm 4.2.1.1 and Algorithm 4.2.2.2 (TFEFF) based on BU-stratified structure and EFF's. Network consists of four cells and 54 driver FET's.

5. TRANSFORMATIONS BASED ON CONFIGURATION CONSIDERATIONS TO CONTROL MOS CELL COMPLEXITIES

While many MOS network synthesis procedures in the literature produce 'minimal' (e.g., [IM 71], [Iba 71], [NTK 72], [Liu 72], [Shi 72], [Lai 76], [Yam 76]) or 'irredundant' (e.g., [Pai 73], [Lai 76],[Yam 76]) networks, the MOS cells of the synthesized networks may frequently be too large for practical implementation. Transient response of a cell or cell layout area, or both, may be unacceptable due to an excessive number of FET's connected in series or parallel in the cell's driver.[Spe 69]

Although at least one paper, [Liu 72], attempts to deal with this problem (see Liu's scheme (2), page 155, and Algorithm 5.4.1, p. 144, of [Liu 72]), the proposed methods - one of which maintains cell minimality - are not designed to meet any specific limitations on the number of FET's in series or parallel in a cell's driver. Thus, after synthesis, the generated network may still be inappropriate for practical implementation.

This section will offer relatively simple transformations which may be performed on MOS networks, taking into consideration a network's configuration only. These may be used to render the networks synthesized by more sophisticated methods into practicable configurations. Or, they can simply be used to 'massage' any MOS design in an attempt to satisfy some desired creteria which might be difficult or inconvenient to obtain by computer programming.

Since the primary reason for presenting these transformations is their utility in obtaining practical networks, extensions to more powerful versions involving the consideration of more complex factors (e.g., the functions realized by the cells) will be avoided so that the transformations themselves do not become impractical. Computer program implementations, however, could probably benefit by employing more sophisticated extensions of these transformations.

Furthermore, a discussion of a systematic application of the transformations proposed in this section will be omitted since the approach taken is strongly dependent on the result sought, and, also, any general approach which may be specified would likely not yield the best results in many cases.

The transformations to be presented are divided into the following five classes: logic insertion/extraction, logic duplication/combination, logic integration/distribution, redundant logic addition/deletion, logic factorization/defactorization, to be discussed in Sections 5.1 through 5.5, respectively. The last of these classes consists of strictly intracell transformations. The next-to-last class, redundant logic addition/deletion, is also made up of intracell transformations, but these are based partly on considerations of connections outside the transformed cell. The reason for class names being of the form 'logic P/Q' is that all of the transformations are presented in complementary pairs - transformation in one direction being designated 'P', and in the other, 'Q'.

To facilitate the illustration of transformations in the next several sections, let the following symbolic conventions be established: A figure such as Fig. 5.1 should be understood to represent an MOS cell with an unspecified or arbitrary driver. In general, let the symbol in Fig. 5.2 represent any two-terminal (terminals labelled a and b in this example) network of FET's. Fig. 5.3 is an example of a partially specified driver and set of inputs. It is to be understood that the specified portion is only connected to the remainder of the driver through terminals a and b.

The following definitions will also be necessary for subsequent discussion.

Definition 5.1 The S-deletion of a two-terminal subnetwork of FET's from the driver of an MOS cell is the replacement of this subnetwork by a short-circuit (i.e., a direct connection) between its two terminals. The O-deletion of a two-terminal subnetwork of FET's from the driver of an MOS cell is the replacement of this subnetwork by an open-circuit (i.e., no connection) between its two terminals.

Fig. 5.4 gives examples of S-deletion and O-deletion. Fig. 5.4(a) shows an MOS cell with a two-terminal subnetwork of FET's encircled. The S-deletion of this subnetwork results in Fig. 5.4.(b). The O-deletion gives the MOS cell of Fig. 5.4(c).

Definition 5.2 The S-addition of a two-terminal subnetwork of FET's to the driver of an MOS cell is the replacement of a short-circuit (i.e., a direct connection) between two terminals by this subnetwork. The O-addition of a two-terminal subnetwork of FET's to the driver of an MOS cell is the connection of this subnetwork between any two existing terminals in the driver.

Examples of S-addition and O-addition are given in Fig. 5.5. The original MOS cell and two-terminal subnetwork of FET's to be added are shown in Fig. 5.5(a). One possible S-addition of the subnetwork to the cell's driver is shown in Fig. 5.5(b); one possible O-addition is given in Fig. 5.5(c)(S- and O-additions may also be made in several other locations in this MOS cell).

While the presentation in this section will generally be made with series-parallel FET networks in mind, the two preceding definitions also apply to

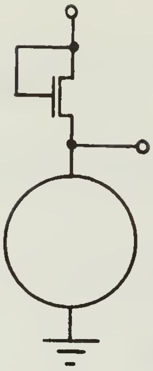


Fig. 5.1 MOS cell with unspecified or arbitrary driver.

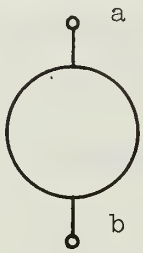


Fig. 5.2 Symbol for unspecified or arbitrary FET subnetwork.

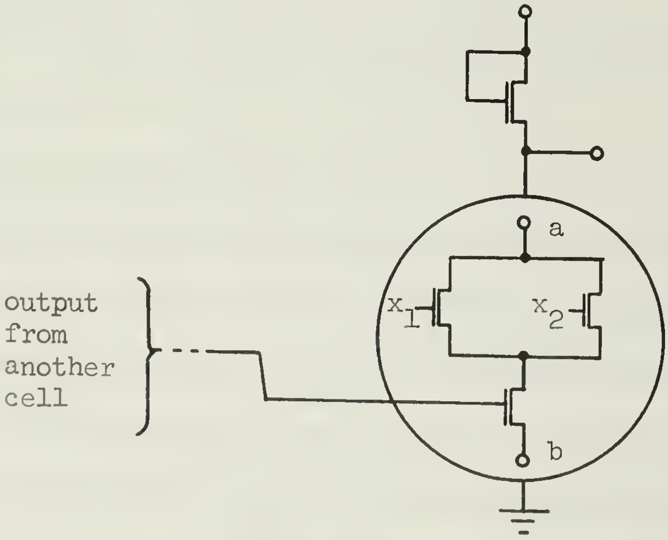
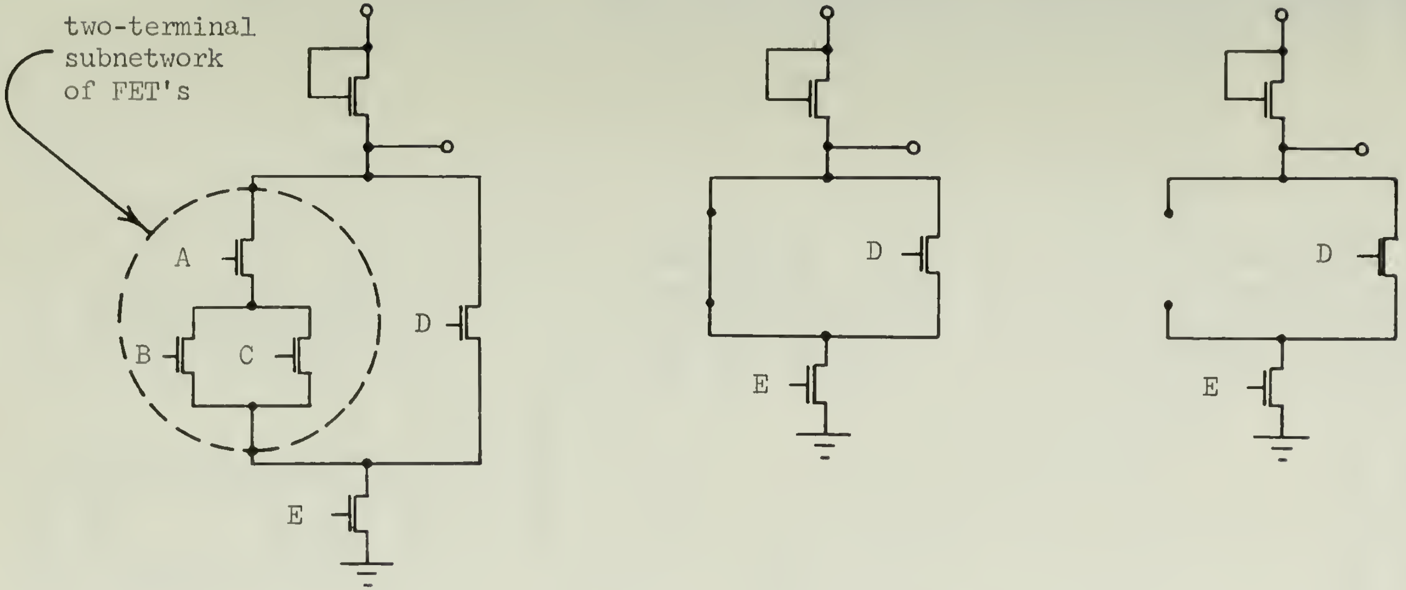


Fig. 5.3 MOS cell with partially specified driver.

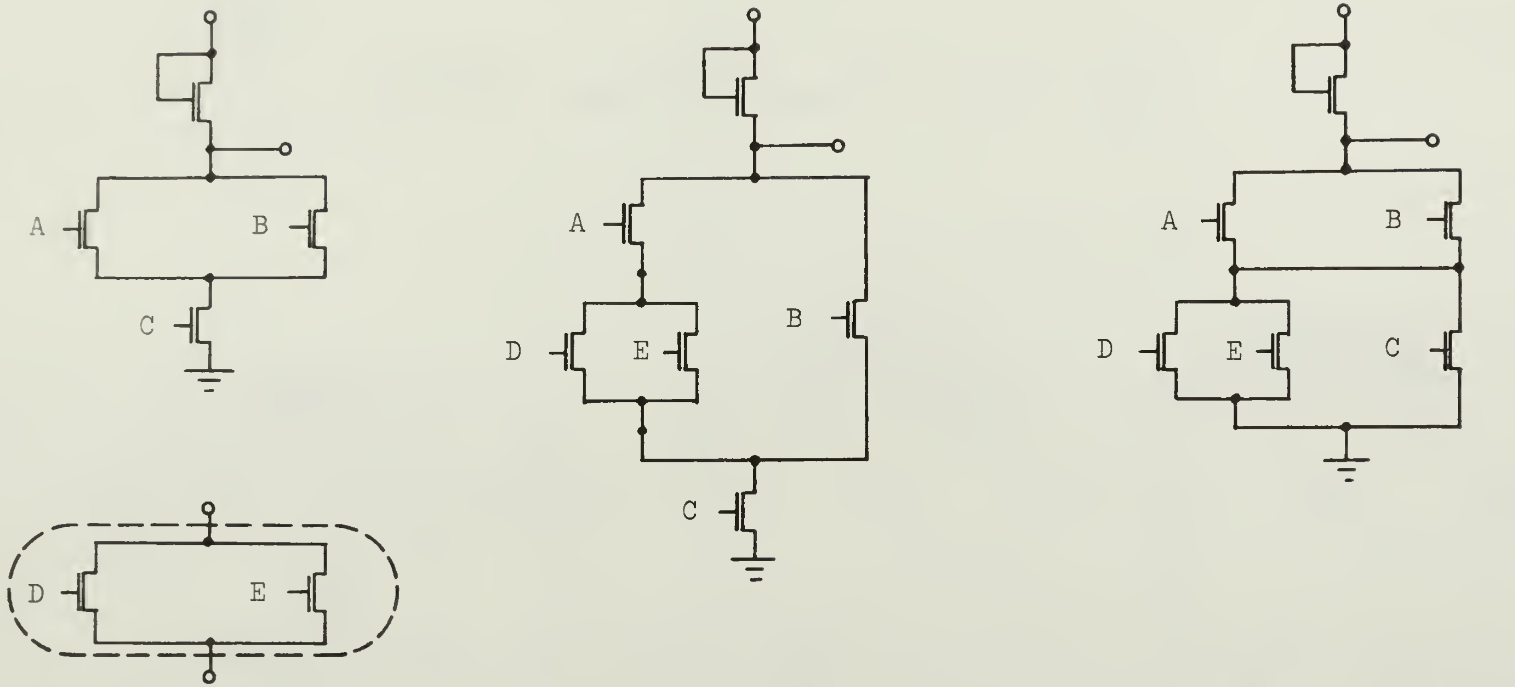


(a) Original MOS cell.

(b) MOS cell of (a) after S-deletion of two-terminal subnetwork.

(c) MOS cell of (a) after O-deletion of two-terminal subnetwork.

Fig. 5.4 Examples of S-deletion and O-deletion.



(a) Original MOS cell and two-terminal FET sub-network to be added.

(b) MOS cell of (a) after S-addition of two-terminal subnetwork.

(c) MOS cell of (a) after O-addition of two-terminal subnetwork.

Fig. 5.5 Examples of S-addition and O-addition.

networks containing'bridge'connections (as will the transformations involving these definitions which will be given later).

Some of the transformations to be given involve the concept of a dual network. The dual of a two-terminal (planar) network of FET's can be obtained by determining the dual (as defined in graph theory) of the network considered as a two-terminal graph.

5.1 Logic Insertion/Extraction

This class of transformations consists of three pairs of transformations - each pair being a transformation and its inverse.

Fig. 5.1.1 shows the first pair of tansformations. As in all of the transformations to follow, the network being transformed is assumed to be 'loop-free' or 'feed-forward,' and this property is preserved by the transformations. For convenience, a cell with an output designated g_i will be simply referred to as cell g_i.

The subnetworks N and N' in Fig. 5.1.1 are referred to as such since they may, in general, be just a part of a larger network.

The conditions on cell inputs and outputs are as indicated in the network diagrams: Cell g_i may be an input to cells other than g_j, including any of $g_{k_1},\ldots,g_{k_t}$. (It is also possible that g_i may control FET's in the driver of g_j other than the one shown, but any such FET's are actually redundant and could be eliminated by transformations described in Section 5.5.) Non-output cell g_j (g_j' in N') is an input only to cells $g_{k_1},\ldots, g_{k_t}$, and it may control one or more FET's in each of these cells. Finally, cell g_i in N' may or or may not exist (a fact represented by the encircling dashed line in Fig. 5.1.1) depending on whether or not it has outputs to cells other than cell g_j of N.

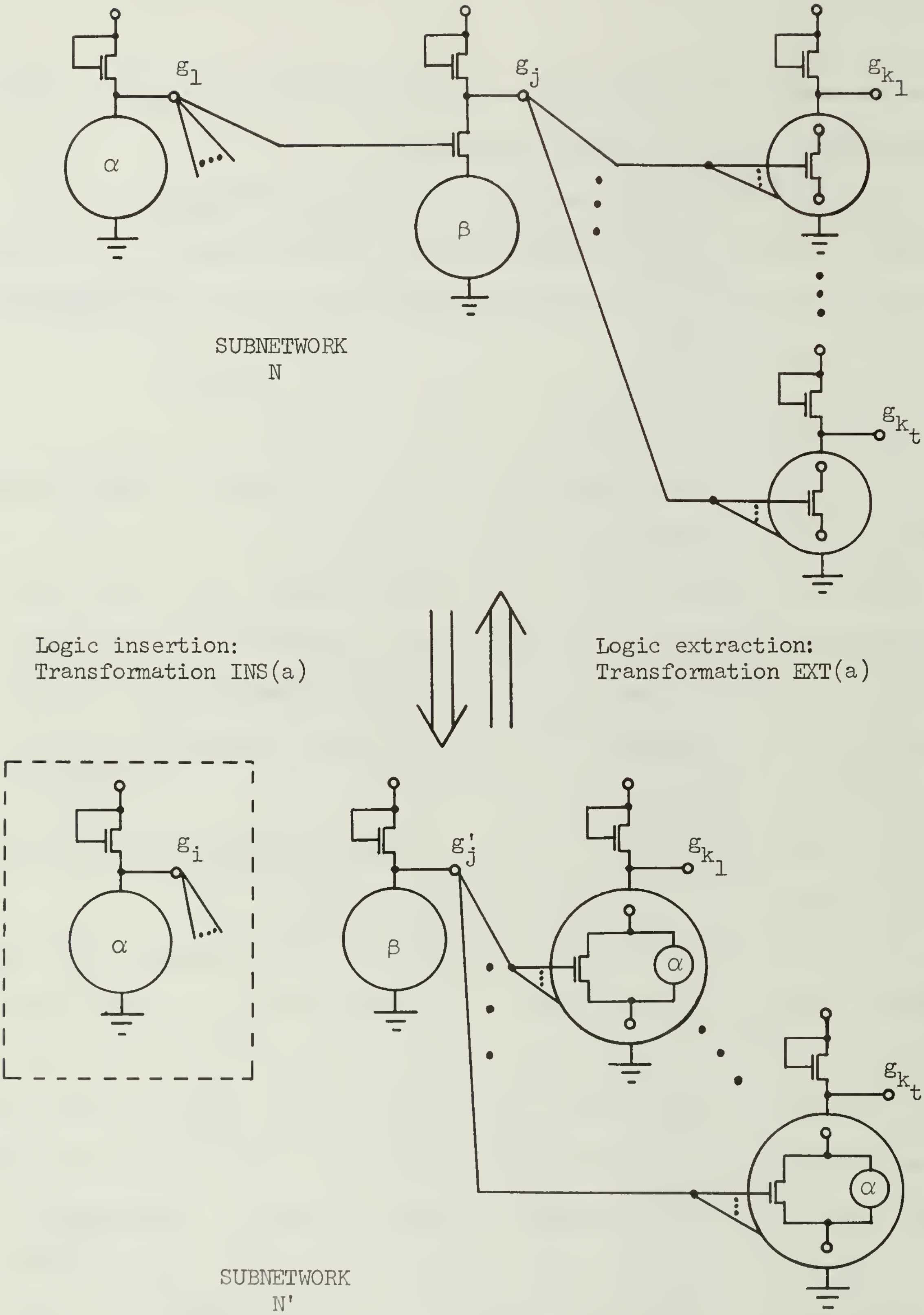


Fig. 5.1.1 First pair of logic insertion/extraction transformations: INS(a) and EXT(a).

The symbols α andβ in Fig. 5.1.1 indicate two-terminal networks of FET's. It is easily seen that for N or N' to satisfy the loop-free condition, FET's controlled by $g_{k_1},\ldots,g_{k_t}$ can not appear in α.

The transformation from subnetwork N to subnetwork N' is classified as 'logic insertion' since the 'logic' realized by the driver of cell g_i, α, is 'inserted' into each of the cells $g_{k_1},\ldots,g_{k_t}$ by the transformation. This particular transformation is designated Transformation INS(a)('INS' being an abbreviation of 'logic insertion' and the '(a)' being added to differentiate it from a second transformation of the same type which will be introduced shortly).

Transformation INS(a) When a cell g_i controls at least one FET along every path through the driver of a second, non-output cell, g_j, then: (1) every FET controlled by the output of g_j can be replaced by a parallel connection of that FET and a two-terminal network of FET's in the same configuration as the driver of g_i and (2) all FET's controlled by g_i in the driver of g_j can be S-deleted.

Theorem 5.1.1 The transformation of a subnetwork of a given network by INS(a) does not alter the network's outputs.

Proof For each possible input vector, A, one of the following two cases must occur. Let N and N' denote the altered subnetwork before and after transformation, respectively. Let g_j' denote the function of cell g_j after the S-deletion of all of the FET's controlled by g_i in its driver.

Case 1 There is a conducting path through the driver of g_i for input vector A. Thus, in N, $g_i(A) = 0$, and $g_j(A) = 1$ since at least one FET controlled by g_i appears along each path through the driver of g_j. Hence every FET controlled by g_j in an immediate successor is conducting. In N', the parallel configuration substituted for every FET controlled by g_j in N is also conducting for A. Thus, the outputs of g_i and every immediate successor - and therefore every successor (including any output cells) - of g_j remain unchanged for A.

Case 2 No conducting path exists through the driver of g_i for input vector A. Thus, in N, $g_i(A) = 1$, and g_j with input $g_i = 1$ is obviously functionally equivalent to g_j with FET's controlled by g_i S-deleted, i.e., $g_j(A) = g_j'(A)$. Thus, $g_j'(A)$ is the transmission function realized in N by every FET controlled by cell g_j in the driver of an immediate successor. In N', the parallel configuration substituted for every FET controlled by g_j in N has no conducting path through the portion corresponding to the driver of g_i. Thus the transmission function corresponds to $g_j'(A)$ as in the case of N. Hence, for this case also the outputs of g_i and every immediate successor and successor of g_j remain unchanged for A.

It is clear that the functions of any output cells which are not successors of g_j can not be affected by the transformation.

Q.E.D.

The inverse of Transformation INS(a) is Transformation EXT(a). It transforms subnetwork N' of Fig. 5.1.1 into subnetwork N and is classified as 'logic extraction' since 'logic' realized by the FET configuration α in the drivers of cells $g_{k_1},\dots,g_{k_t}$ is 'extracted' by the transformation.

Transformation EXT(a) Consider a non-output cell g_j' and a two-terminal FET subnetwork of configuration α and transmission function $\bar{g}_i$. Further suppose that none of the FET's in the subnetwork are controlled by cell g_j'. If every FET controlled by g_j' is in parallel with a subnetwork of configuration α, then this parallel configuration can be replaced in every case by a single FET controlled by any cell g_j obtained as follows: (1) if no cell g_i having a driver of configuration α currently exists in the network, one is created; (2) a cell g_j is then obtained from cell g'_j by the S-addition of at least one FET controlled by g_i along each path through the driver of g'_j.

Theorem 5.1.2 The transformation of a subnetwork of a given network by EXT(a) does not alter the network's outputs.

Proof Let N' and N denote the altered subnetwork before and after Transformation EXT(a), respectively. By Transformation INS(a), N can be transformed to a network N" while maintaining the outputs of N. By the definitions of the two transformations, network N" can be chosen to be identical to N' (while this is always true, for some subnetworks N, it is also possible to obtain one or more N" different from N' through INS(a)). By Theorem 5.1.1, N and N", and therefore N and N', must have identical outputs.

Q.E.D.

It should be noted that the proofs of Transformations INS(a) and EXT(a) do not actually depend on the identical FET network α being in both the driver of g_i and the drivers of $g_{k_1},\ldots,g_{k_t}$. Actually it is sufficient for physically different α subnetworks to have identical transmission functions rather than being required to have identical structures. However, so that the transformations discussed in this section, Section 5, remain simple enough to be applied by hand, it is desired that they depend only upon consideration of configuration and not upon analysis of realized functions. It may be apparent to the reader that if more sophisticated considerations based on function were permitted, restrictions on the different subnetworks designated α in Fig. 5.1.1 may be relaxed still further until even identical functions need not be required u under appropriate circumstances.

The second pair of logic insertion/extraction transformations is illustrated in Fig. 5.1.2. The transformations of this pair are designated INS(b) and EXT(b), and they are quite similar to INS(a) and EXT(a), respectively. Whereas the driver of g_j in Fig. 5.1.1 contains, in series with FET network β, an FET controlled by g_i, the driver of g_j in Fig. 5.1.2 has an FET controlled by g_i in parallel with FET network β. Also, the 'inserted' logic, FET network α, is placed in series with each FET controlled by g_j rather than in parallel as in the previous case. Again, for the purposes of Transformations INS(b) and EXT(b), the FET network β need not be free of FET's controlled by g_i, although such FET's can easily be shown to be redundant.

The two transformations of this second pair are given as follows:

Transformation INS(b) When a cell g_i controls one or more FET's in a second, non-output cell g_j, at least one of which is the sole FET along a path through the driver of g_i, then: (1) every FET controlled by the output of g_j can be replaced by a serial connection of that FET and a two-terminal network of FET's in the same configuration as the driver of g_i and (2) all FET's controlled by g_i in the driver of g_j can be 0-deleted.

Transformation EXT(b) Consider a non-output cell g_j' and a two-terminal FET subnetwork of configuration α and transmission function $\overline{g}_i$. Further suppose that none of the FET's in the subnetwork are controlled by cell g_j'. If every FET controlled by g_j' is in series with a subnetwork of configuration α, then this serial configuration can be replaced in every case by a single FET controlled by any cell g_j obtained as follows: (1) if no cell g_i having a driver of configuration α currently exists in the network, one is created; (2) a cell g_j is then obtained from cell g_j' by the 0-addition of FET's controlled by g_i to its driver - at least one of which is added in parallel to the existing driver.

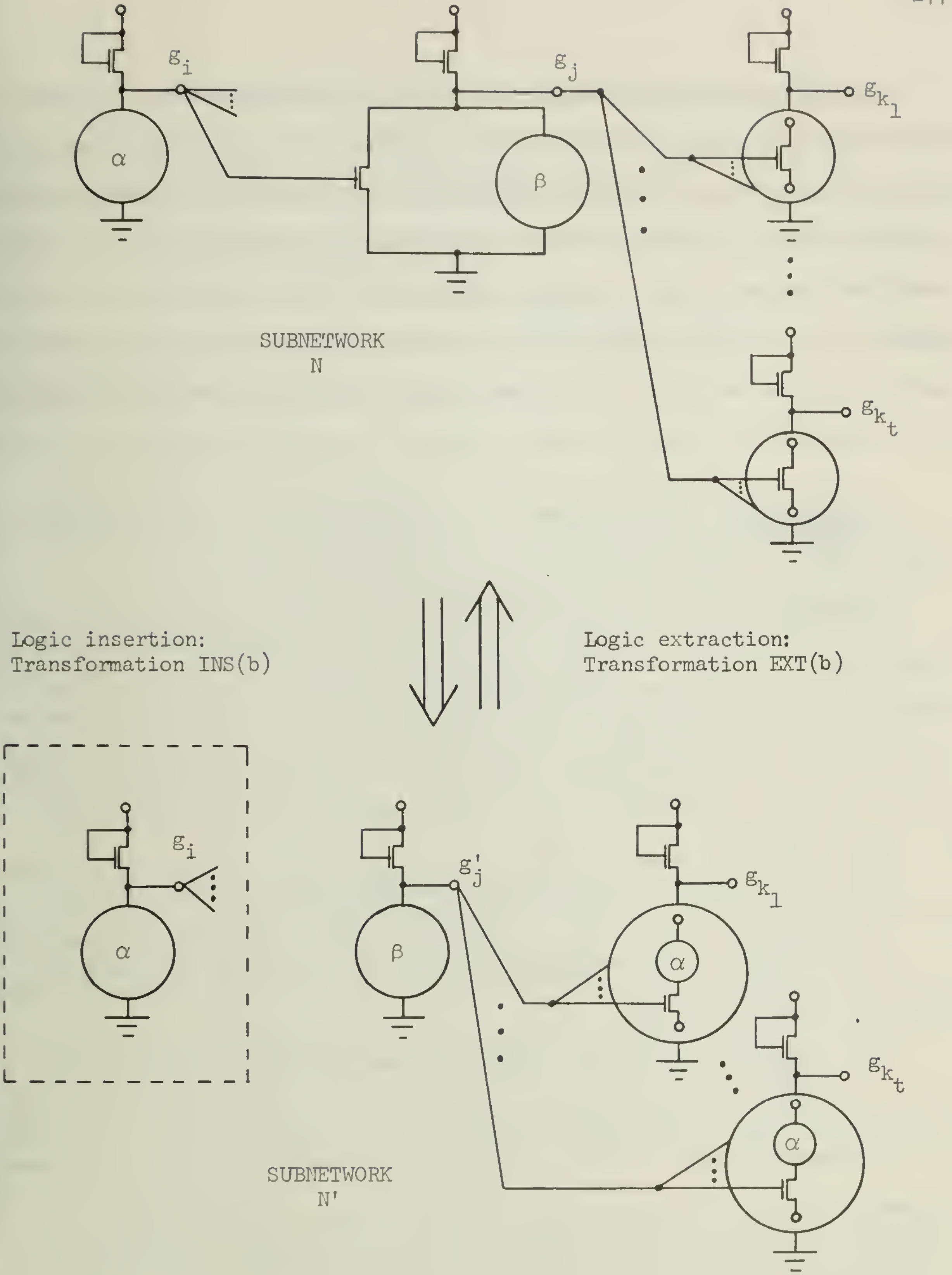

g_i
g_j
g_k1
α
β
g_kt
SUBNETWORK N
Logic insertion: Transformation INS(b)
Logic extraction: Transformation EXT(b)
g_i
α
g'_j
β
g_k1
α
g_kt
α
SUBNETWORK N'

Fig. 5.1.2 Second pair of logic insertion/extraction transformations: INS(b) and EXT(b).

It might be noted that the condition in both Transformation EXT(a) and EXT(b) prohibiting the existence of FET's controlled by g_j' in α is actually redundant under the practical assumption that the discussion is limited to MOS cells with a finite number of FET's. It is fairly obvious that to have an FET controlled by g_j' in a subnetwork of configuration α and to still satisfy the conditions of the logic extraction transformations which require every FET controlled by g_j' to be in series or parallel with a subnetwork of configuration α, would necessitate an infinite number of FET's in configuration α.

Theorem 5.1.3 The transformation of a subnetwork of a given network by either INS(b) or EXT(b) does not alter the network's outputs.

Proof Let N and N' denote the altered subnetwork before and after Transformation INS(b) (after and before Transformation EXT(b)), respectively. It is sufficient to show: (1) that N' can be obtained by INS(b) from N if N can be obtained from N' by EXT(b) and (2) that INS(b) preserves the outputs of cells $g_i, g_{k_1}, \ldots, g_{k_t}$ for every input vector. From this, it can be deduced that both INS(b) and EXT(b) preserve outputs of cells $g_i, g_{k_1}, \ldots, g_{k_t}$, and as these are the only possible cells connected from subnetwork N (N') to the rest of the network, the outputs of all other cells outside of N (N') are also preserved - hence the theorem statement.

By a comparison, INS(b) and EXT(b) can easily be seen to be inverse transformations; consequently (1) is true.

To demonstrate (2), consider each possible input vector A to the network. For each A one of the following two cases must occur. (Let g_j' denote the function of cell g_j after the 0-deletion of all of the FET's controlled by g_i in its driver.)

Case 1 There is no conducting path through the driver of g_i (FET subnetwork α) for input vector A. Thus, in N, $g_i(A) = 1$, and obviously $g_j(A) = 0$. Hence every FET controlled by g_j in an immediate successor is non-conducting. In N', α in the serial configuration substituted for every FET controlled by g_j in N is also non-conducting. Thus the outputs of $g_i, g_{k_1}, \ldots, g_{k_t}$ remain unchanged from N to N' for A. (Any change in the output of g_j is unimportant since it has no immediate successors outside of the subnetwork nor does it realize a network output function.)

Case 2 There is a conducting path through the driver of g_i, α, for A. Thus, in N, $g_i(A) = 0$, and g_j with input $g_i = 0$ is obviously functionally equivalent to g_j with FET's controlled by g_i 0-deleted, i.e., $g_j(A) = g_j'(A)$. Since there is a conducting path through α for A, it is easy to see that the transmission function realized in N by every FET controlled by g_j is the same as that realized by every substituted serial configuration in N' for A. Hence, for this case also, $g_i, g_{k_1}, \ldots, g_{k_t}$ remain unchanged from N to N' for A.

Q.E.D.

The third pair of transformations is illustrated in Fig. 5.1.3. These transformations, designated INS(c) and EXT(c), can be viewed as essentially degenerate cases of the preceding insertion and extraction transformations, although this is not formally true due to the non-existence of FET's controlled by g_j' in cells $g_{k_1}, \ldots, g_{k_t}$. In this pair of transformations, α again represents the FET subnetwork being inserted or extracted. The transformations are defined as follows:

Transformation INS(c) When a cell g_i controls the sole FET in the driver of a second cell g_j, then: (1) every FET controlled by g_j can be replaced by

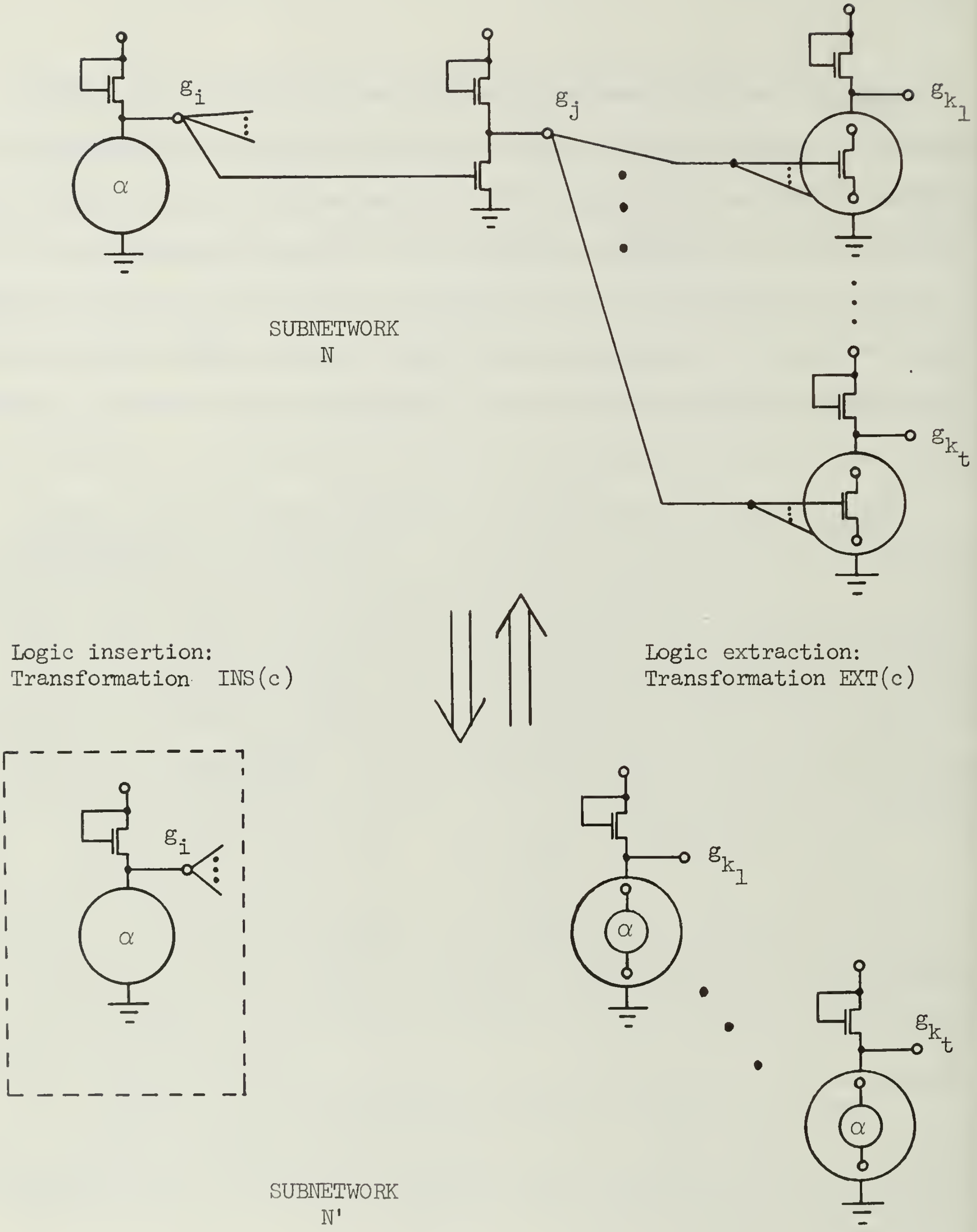


Fig. 5.1.3 Third pair of logic insertion/extraction transformations: INS(c) and EXT(c).

a two-terminal network of FET's, α, in the same configuration as the driver of g_i and (2) if g_j is not an output cell, cell g_j can be removed from the network along with its input connection.

Transformation EXT(c) When one or more cells, $g_{k_1},\ldots,g_{k_t}$, of a network contain a common two-terminal FET subnetwork α realizing transmission function $\bar{g}_i$, α can be replaced in every case by a single FET controlled by an MOS cell, g_j, whose driver consists of a single FET controlled by cell g_i with driver configuration α.

Theorem 5.1.4 The transformation of a subnetwork of a given network by either INS(c) or EXT(c) does not alter the network's outputs.

Proof Let N and N' denote the altered subnetwork before and after Transformation INS(c) (after and before Transformation EXT(c)), respectively. The function realized by cell g_i is just the complement of the transmission function realized by FET network α. Thus the function realized by cell g_j is identical to the transmission function of α and so is the transmission function of every FET controlled by g_j. Clearly then, the outputs of cells $g_i,g_{k_1},\ldots,g_{k_t}$ are unaffected by the transformations. Thus the outputs of all cells outside the subnetwork are also unaffected.

Q.E.D.

Some of the possible uses of logic insertion in general are: to reduce the number of cells in a network, to reduce the number of levels of cells between network inputs and outputs, and/or to prepare for further transformations. Possible uses of logic extraction include: the reduction of the total number of FET's in the network (especially if g_i already exists elsewhere in the network), the simplification of overly complex drivers, and the alteration of the network configuration in preparation for further transformations.

Fig. 5.1.4(a), (b), and (c) demonstrate logic insertion/extraction. A three level network (a) of three cells, g_1, g_2, and g_3 employing 1, 7, and 1 driver FET's respectively, is transformed into a three-cell, two-level network with 9 driver FET's evenly distributed among the cells. This is accomplished by first extracting by Transformation EXT(b) a subnetwork α, consisting of the parallel connection of three FET's controlled by x_2, x_3, and x_4, from g_2 to form the driver of a new cell g_4 (Fig. 5.1.4(b)). Reducing this four-cell network to a three-cell network, Transformation INS(b) inserts a parallel connection of two FET's controlled by x_1 and cell g_4 into cell g_3 in series with the FET controlled by g'_2. The final network is shown in Fig. 5.1.4(c).

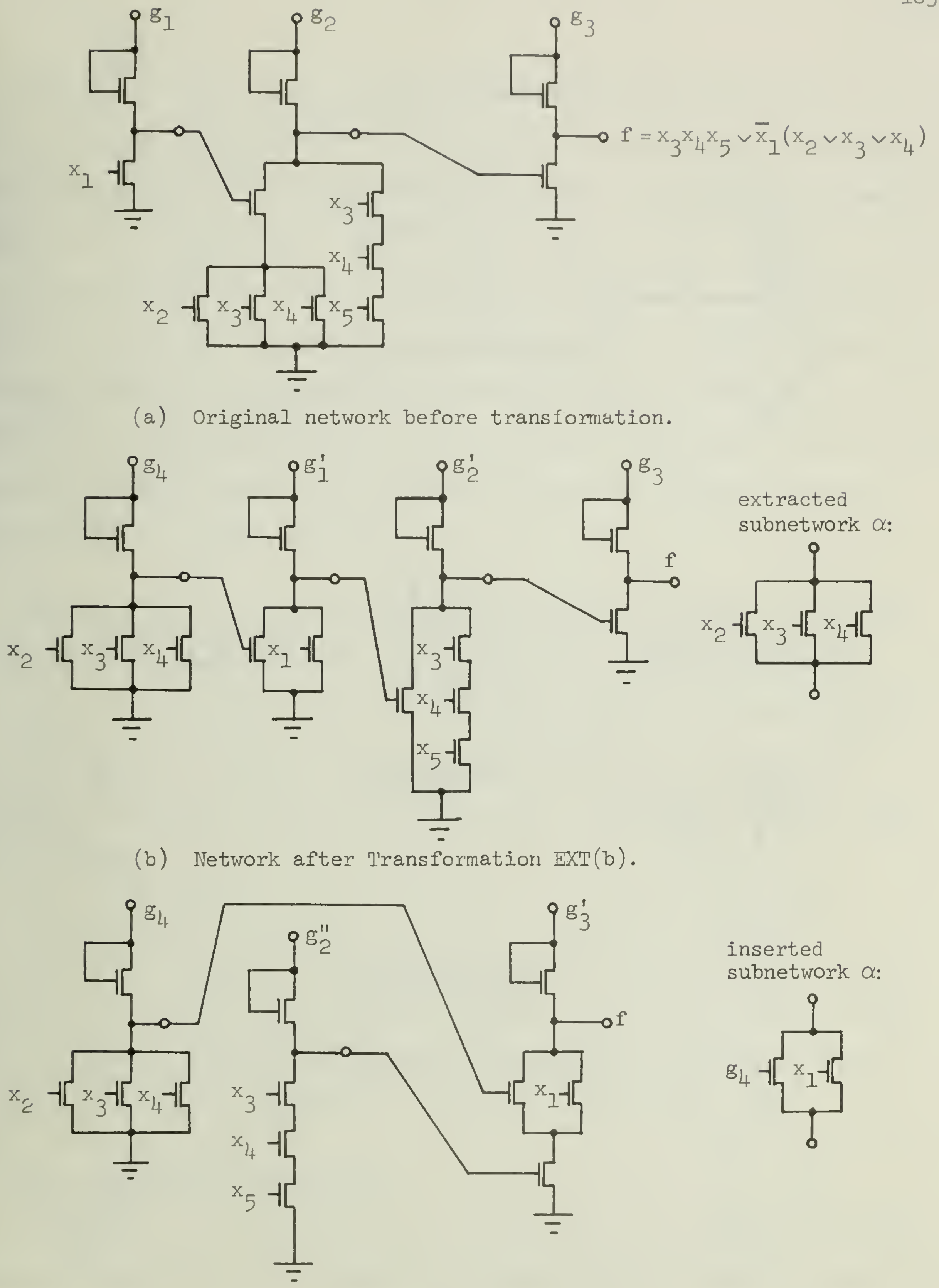
5.2 Logic Duplication/Combination

This class of transformations consists of the single pair of transformations illustrated in Fig. 5.2.1. These transformation are quite simple and are probably most useful prior to or subsequent to other transformations.

The conditions on subnetwork N in Fig. 5.2.1 are as follows: $g_i^1, \ldots, g_i^r$ are a set of cells with identically configured drivers and $g_{k_1}, \ldots, g_{k_t}$ are the set of cells having at least one input from the set $g_i^1, \ldots, g_i^r$. In subnetwork N', $g_{k_1}, \ldots, g_{k_t}$ constitute the set of immediate successors of cell g_i.

The transformations, designated DUP for logic duplication and COM for logic combination, are described as follows:

Transformation DUP A cell g_i with immediate succesors $g_{k_1}, \ldots, g_{k_t}$ can be replaced by a set of (duplicated) cells, $g_i^1, \ldots, g_i^r$, with driver configurations identical to that of g_i if each FET formerly controlled by g_i



(a) Original network before transformation.

(b) Network after Transformation EXT(b).

(c) Network after Transformation INS(b).

Fig. 5.1.4 Demonstration of logic insertion/extraction.

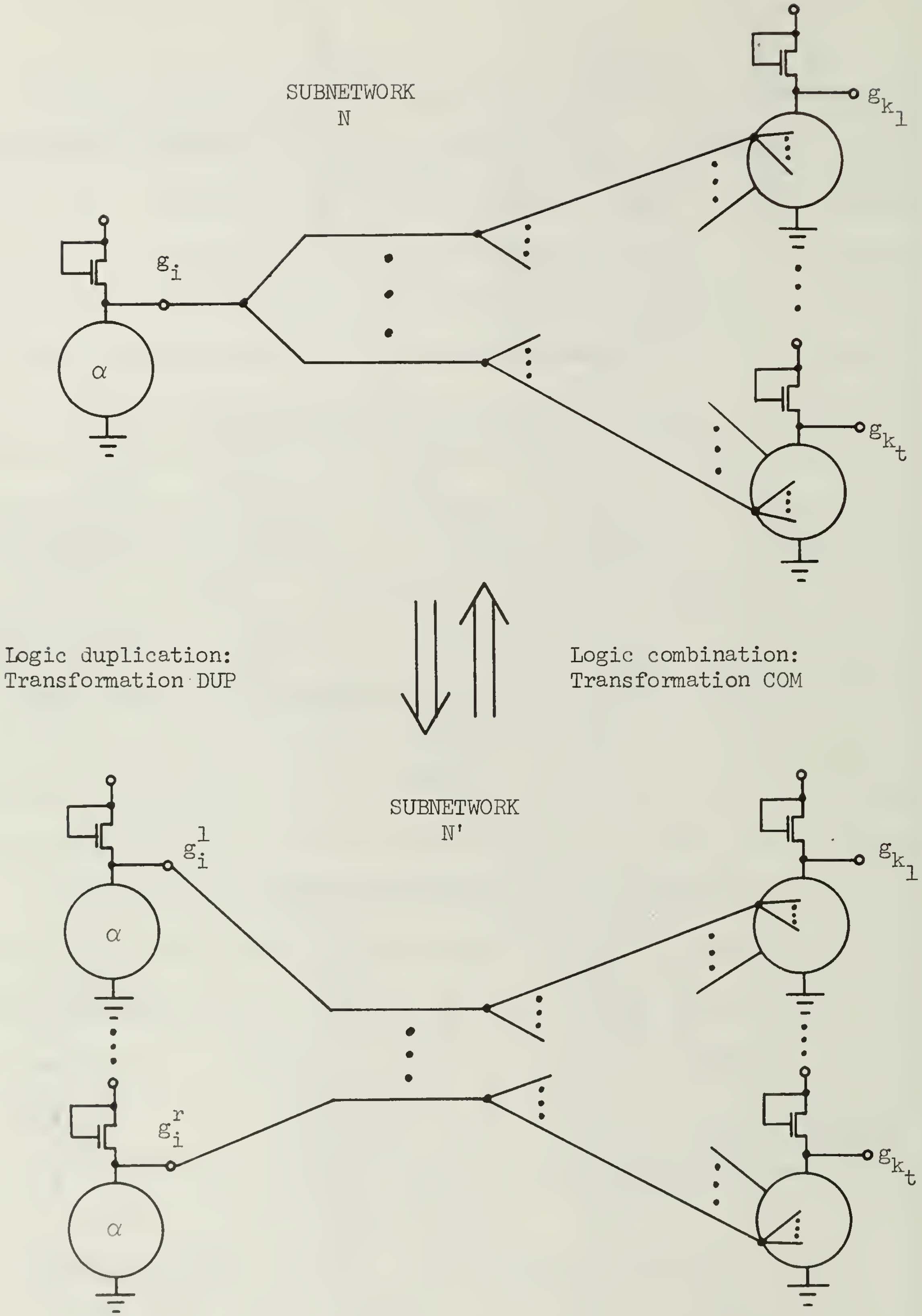


Fig. 5.2.1 Pair of logic duplication/combination transformations: DUP and COM.

in the drivers of $g_{k_1},\ldots,g_{k_t}$ is replaced with an FET controlled by one of the cells in the replacement set. If g_i is an output cell of the network, one or more g_i^j, $1 \leq j \leq r$, is regarded as an output cell realizing the same function as the replaced g_i.

Transformation COM A set of cells $g_i^1,\ldots,g_i^r$ with drivers of identical configuration, α, can be replaced by a single cell, g_i, also having driver configuration α, if each FET formerly controlled by $g_i^1,\ldots,$ or g_i^r is replaced with an FET controlled by g_i. If one or more of the g_i^j, $1 \leq j \leq r$, is an output cell of the network, g_i is regarded as an output cell realizing the same function as the replaced g_i^j.

Theorem 5.2.1 The transformation of a subnetwork of a given network by either Transformation DUP or COM does not alter the network's outputs.

Proof Since $g_i, g_i^1,\ldots,g_i^r$ all have identical drivers, they must realize the same function. Obviously, their outputs can be interchanged without affecting the outputs of any of their successors - including the output cells of the network.

Q.E.D.

The most obvious use for Transformation COM is in the reduction of the number of cells or FET's of a network. Possible uses for DUP include obtaining a network configuration suitable for the application of other transformations or reducing the fan-out of a particular cell.

5.3 Logic Integration/Distribution

Three pairs of transformations are discussed for this class. Two of the pairs are basic and are combined into the generalized third pair of transformations. Unlike the preceding transformations, logic integration creates a new cell by 'integrating' the drivers of the cells it replaces into the larger driver of the new cell. Logic distribution, the reverse process, breaks apart a single cell by 'distributing' portions of its driver to form the drivers of several new replacement cells.

The first basic pair of logic integration/distribution transformations is shown in Fig. 5.3.1. The transformation from subnetwork N to N' is called MAND (for merge AND) since the series, or 'AND' connection, of FET's shown in the driver of cell g_k of N is 'merged' into a single FET in g_k of N'. Similarly, the transformation from N' to N is called DAND (for divided AND) since the single FET shown in the driver of g_k in N' is, in a sense, 'divided' into the several FET's 'ANDed' together in g_k of N.

The second pair of basic transformations MOR (for merge OR) and DOR (divide OR) are illustrated in Fig. 5.3.2 and can be seen to be very similar to MAND and DAND. The FET's formerly in series in the driver of g_k in subnetwork N are now in parallel. Also, the FET networks $\alpha_1,\dots,\alpha_i$ formerly in parallel in g_j of N' are now in series.

The conditions for the two pairs of transformations are quite simple: $g_1,\dots,g_i$ and g_j must be non-output cells having only g_k as an immediate successor and controlling only those FET's shown in the figures; cell g_k may have inputs other than those shown. It should be noted that transformation DUP can often be helpful in altering a network's configuration to satisfy this condition.

The descriptions of the two pairs of transformations are combined in the following:

Transformation MAND (MOR) When non-output cells $g_1,\dots,g_i$ with driver configurations $\alpha_1,\dots,\alpha_i$, respectively, each control only a single FET and

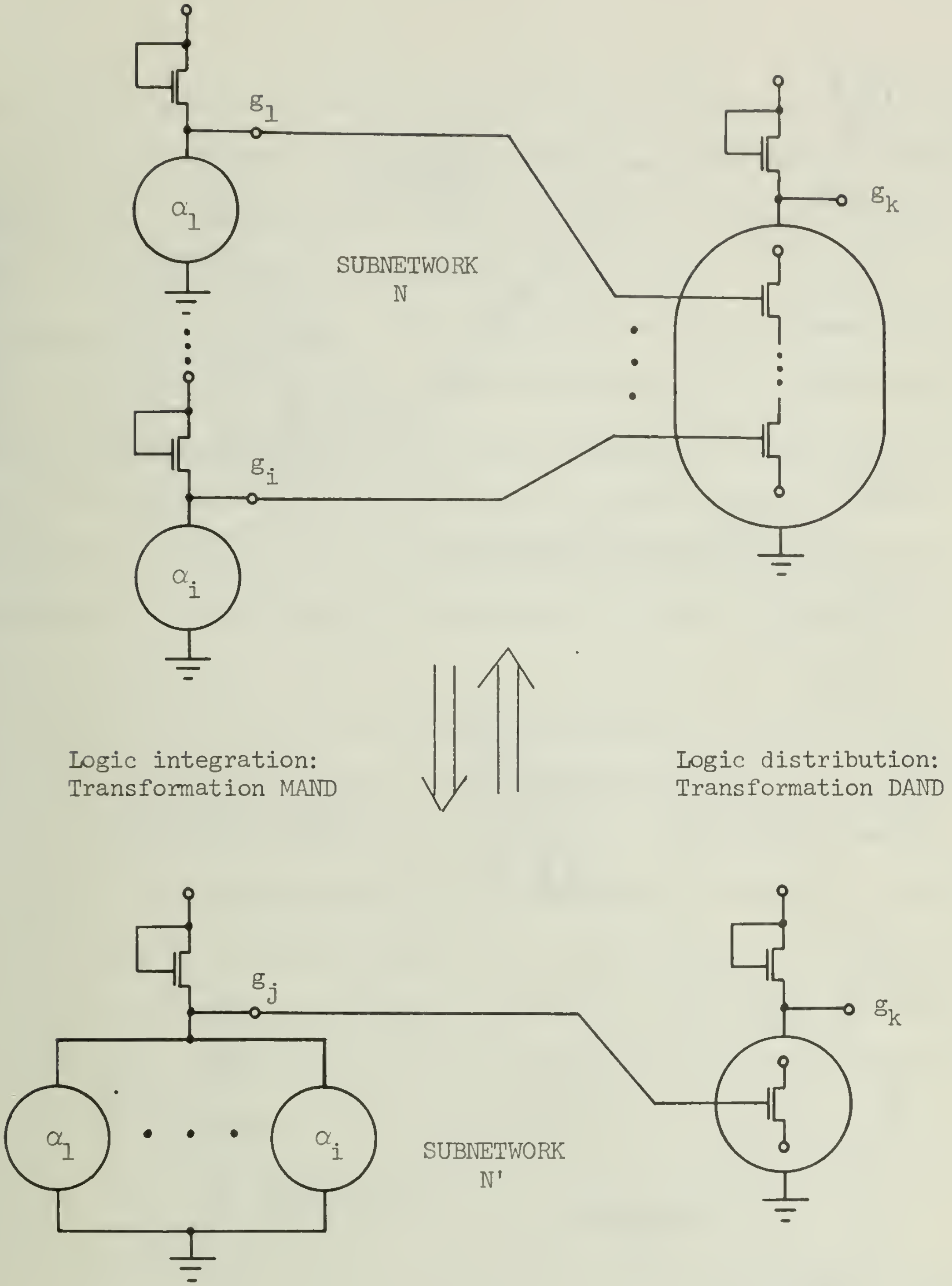


Fig. 5.3.1 First basic pair of logic integration/distribution transformations: MAND and DAND.

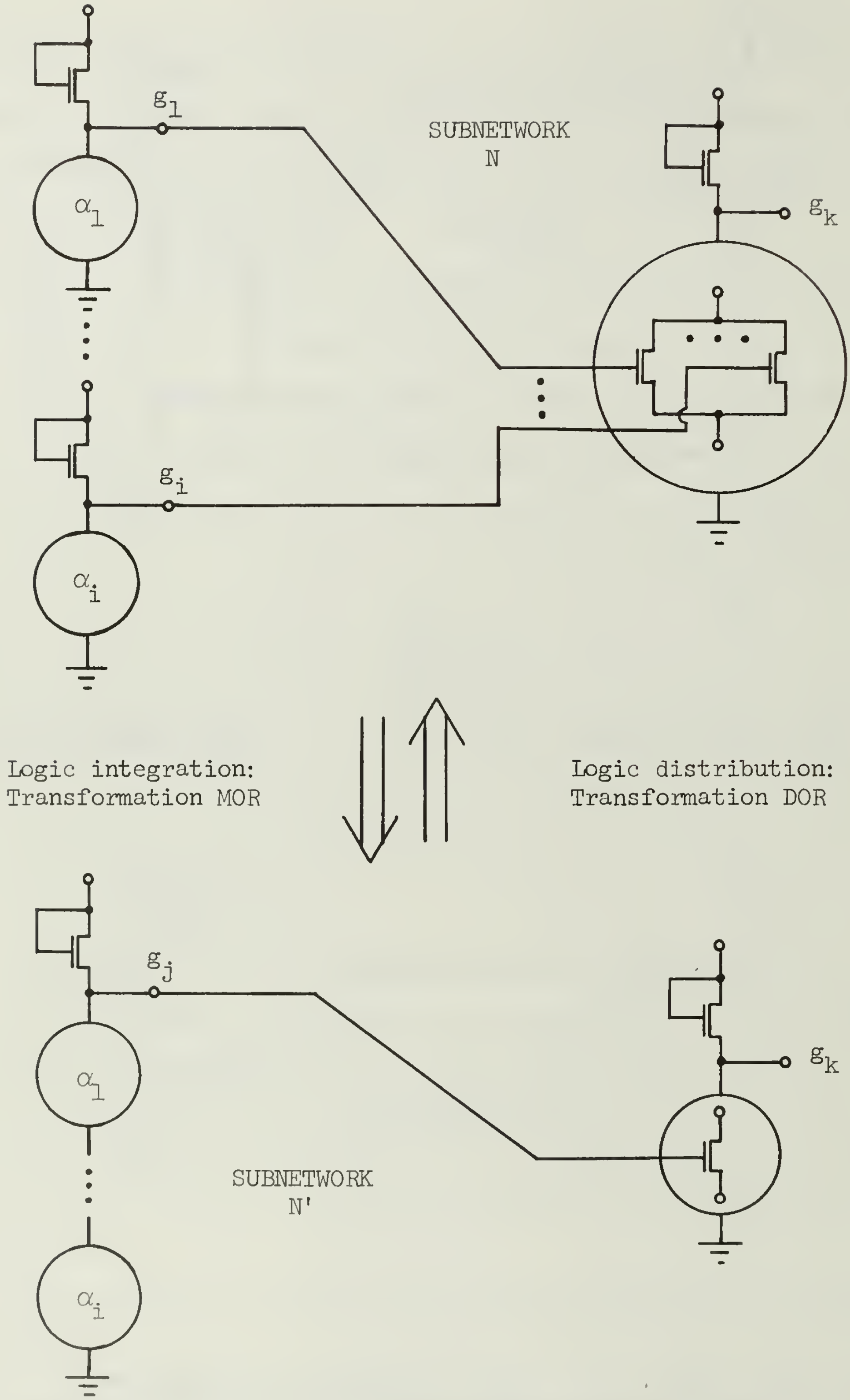


Fig. 5.3.2 Second basic pair of logic integration/distribution transformations: MOR and DOR.

these i FET are connected serially (in parallel) in the driver of a cell g_k, then this series (parallel connection) of FET can be replaced by a single FET controlled by a new cell g_j whose driver is formed by a parallel (serial) connection of the two-terminal FET networks $\alpha_1,\ldots,\alpha_i$.

Transformation DAND (DOR) When a non-output cell g_i controls only a single FET, in a cell g_k, and the configuration of the driver of g_i can be partitioned into i two-terminal FET networks, $\alpha_1,\ldots,\alpha_i$, connected in parallel (series), then g_j can be replaced by i new cells $g_1,\ldots,g_i$ with respective driver configurations $\alpha_1,\ldots,\alpha_i$ such that the single FET formerly controlled by g_j is replaced by a serial (parallel) connection of i FET - one controlled by each of the cells $g_1,\ldots,g_i$.

The proofs that these four transformations preserve network outputs are omitted since the transformations are really special cases of the more general pair of transformations illustrated in Fig. 5.3.3.

The conditions for this pair of transformations are similar to those for the two basic pairs, with the exception that the cells $g_1,\ldots,g_i$ and g_j are no longer constrained to controlling a single FET: $g_1,\ldots,$ and g_i must be non-output cells having only g_k as an immediate successor and controlling only FET's in a FET subnetwork β; β must have inputs only from $g_1,\ldots,$ and g_i; g_j must be a non-output cell with only one output controlling a single FET in the driver of g_k; and cell g_k may have inputs other than those shown.

The following describes the two generalized logic integration/distribution transformations:

Transformation INT Suppose $g_1,\ldots,g_i$ are non-output cells with driver configurations $\alpha_1,\ldots,\alpha_i$, respectively. Further suppose that all of the FET's controlled by $g_1,\ldots,g_i$ constitute a two-terminal FET subnetwork of configuration

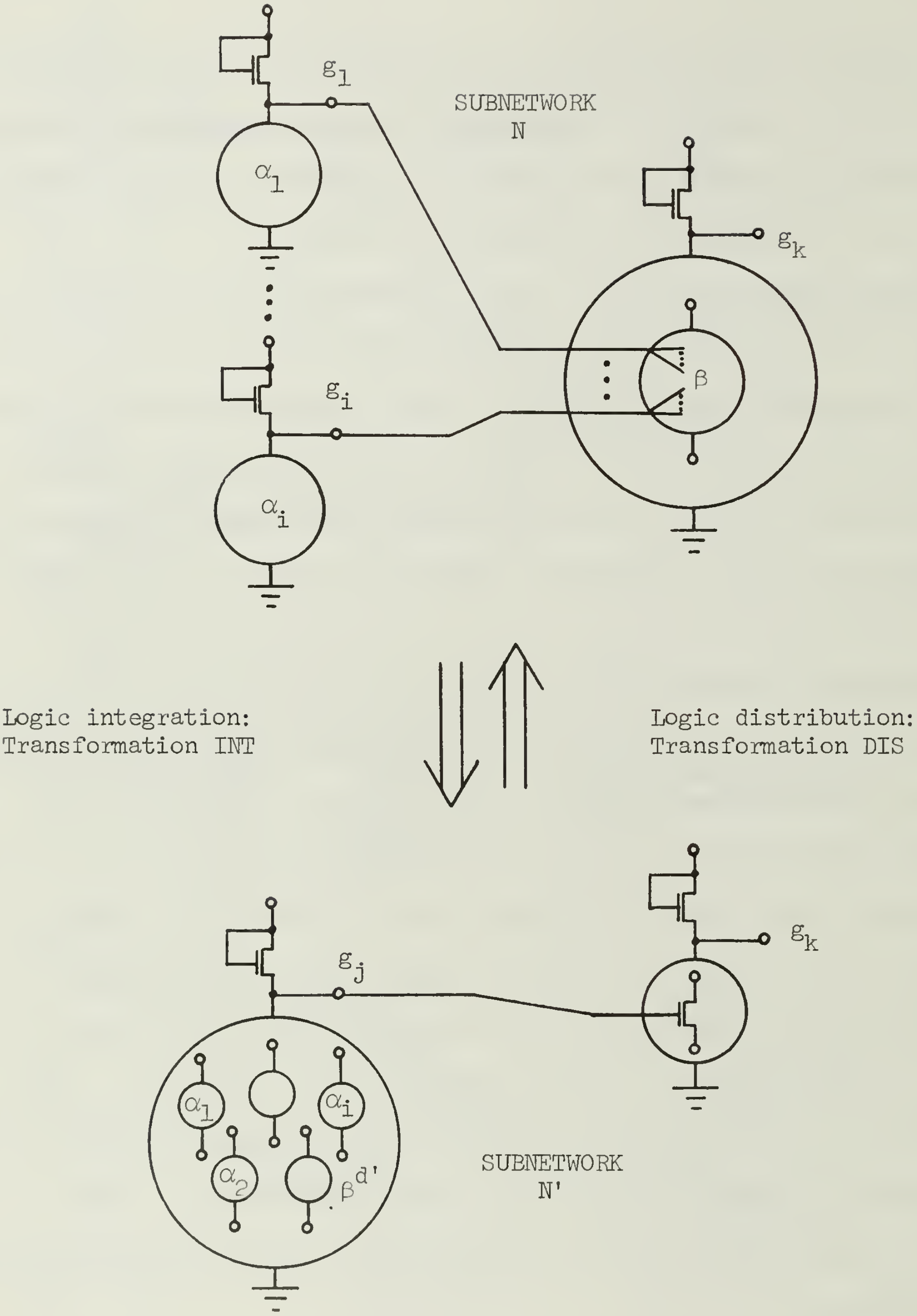


Fig. 5.3.3 General pair of logic integration/distribution transformations: INT and DIS.

β in the driver of a gate g_k. Then, this subnetwork in g_k can be replaced by a single FET connected to the output of a new cell g_j whose driver is constructed as follows: (1) the dual network, β^d, of β is determined; (2) for $\ell = 1,\ldots,i$, each FET in β^d controlled by g_ℓ is replaced by an FET subnetwork of configuration α_ℓ; (3) the resulting FET network, $\beta^{d'}$, is the driver of the new g_j.

Tranformation DIS Suppose a non-output cell g_j has only one output connection, to an FET in the driver of a cell g_k. Further suppose that the FET network constituting the driver of g_j, $\beta^{d'}$, can be partitioned into p two-terminal subnetworks of $i(i \leq p)$ different configurations, $\alpha_1,\ldots,\alpha_i$. Then the single FET in g_k controlled by g_j can be replaced with an FET subnetwork β determined as follows: (1) i new cells $g_1,\ldots,g_i$ are constructed having drivers of configurations $\alpha_1,\ldots,\alpha_i$, respectively; (2) an FET network β^d is then derived from $\beta^{d'}$ by replacing each two-terminal subnetwork of configuration α_ℓ by single FET controlled by g_ℓ, $\ell = 1,\ldots,i$; (3) β is then obtained as the dual network of β^d.

Theorem 5.3.1 The transformation of a subnetwork of a given MOS network by either Transformation INT or DIS does not alter the network's outputs.

Proof First consider Transformation INT.

With $g_1,\ldots,g_i$ designating the functions realized by cells $g_1,\ldots,$ g_i, respectively, let the transmission function of the FET subnetwork β to be replaced be $f^\beta(g_1,\ldots,g_i)$. Then the dual network β^d of β must realize the dual transmission function, $\overline{f}^\beta(\overline{g}_1,\ldots,\overline{g}_i)$, and replacing each FET in β^d connected to the output of g_ℓ with α_ℓ (whose transmission function is obviously $\overline{g}_\ell$) for $\ell = 1,\ldots,i$, to form configuration $\beta^{d'}$ will result in the transmission function $\overline{f}^{\beta}(g_1,\ldots,g_i)$ for the driver of the new cell g_j. Hence, cell g_j will realize the function $f^{\beta}(g_1,\ldots,g_i)$ which will, therefore, also be the transmission function of any FET connected to the output of g_j.

Transformation INT replaces a two-terminal FET subnetwork of the driver of g_k with a single FET of the identical transmission function. Thus the function realized by cell g_k will be unchanged. Since $g_1,\ldots,g_i$ are non-output cells and have no other output connections than to cell g_k, the outputs of any network of which cells $g_1,\ldots,g_i$ and g_k form a subnetwork will have its outputs unaffected by Transformation INT.

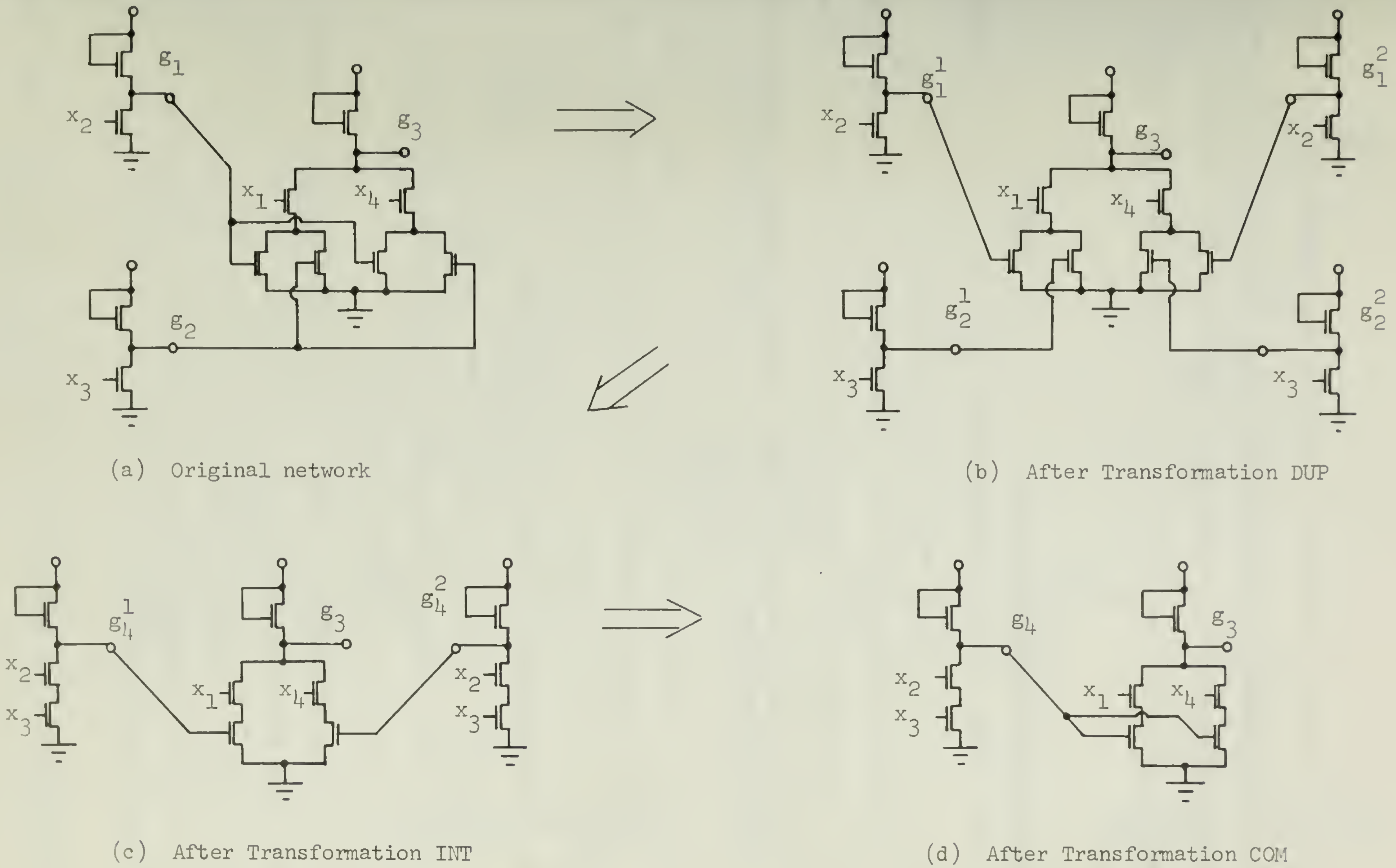
The proof for Transformation DIS is similar but basically proceeds in reverse order.

Q.E.D.

Initially, the pre-transformation conditions given for INT or DUP may appear to be restrictive. For example: $g_1,\ldots,g_i$ and g_j can only have output connections to one other cell; the inputs from $g_1,\ldots,g_i$ to g_k must be the only inputs to FET subnetwork β; cell g_j can control only a single FET in g_k; etc. The appropriate use of logic duplication and/or logic combination, however, can circumvent many of these apparent limitations.

For example, if $g_1,\ldots,g_i$ were connected to two identical FET subnetworks β_1 and β_2 in g_k rather than the single β specified, then: $g_1,\ldots,g_i$ could be duplicated by Transformation DUP; Transformation INT performed based on β_1 (as β) and then on β_2, to obtain g_j^1 and g_j^2 in N', respectively; and, finally, the identical cells g_j^1 and g_j^2 combined into a single g_j by Transformation COM. Fig. 5.3.4 illustrates this strategy for an example in which $i = 2$.

Depending upon the specific configuration of a given network, appropriate uses for logic integration include: a reduction in the number of MOS cells,



(a) Original network

(b) After Transformation DUP

(c) After Transformation INT

(d) After Transformation COM

Fig. 5.3.4 Example of use of logic duplication/combination to enhance the utility of logic integration.

a reduction in the number of FET's or intercell connections, and a reduction in the complexity of the driver of g_k. Similarly, possible uses of logic distribution are in: reducing the number of FET's, reducing the complexity of the driver of g_j, and preparing for a subsequent transformation.

5.4 Redundant Logic Addition/Deletion

For this class, two pairs of transformations will be discussed. Although some similarities can be found between these transformations and those of logic insertion/extraction, this class of transformations differs from those previously given mainly in that FET subnetworks are simply 'added' or 'deleted' (hence, the class name), whereas previously discussed classes of transformations involve 'relocations' of FET subnetworks. In addition, since the FET subnetworks concerned in redundant logic addition/deletion can be added or deleted at will without changing the outputs of the corresponding MOS networks, they can obviously be termed 'redundant.'

Fig. 5.4.1 illustrates the first pair of transformations, RAD(a) and RDE(a) (for redundant logic addition and redundant logic deletion, respectively). The second pair, Transformation RAD(b) and RDE(b) shown in Fig. 5.4.2, can be seen to be basically similar.

The conditions for RAD(a) or RDE(a) (RAD(b) or RDE(b)) are as follows: along every path from a non-output cell g_i to an output cell of the network to which it belongs, there exists a cell such that every FET connected to its output is in series (parallel) with an FET subnetwork of configuration α.

The four transformations are expressed more completely as follows:

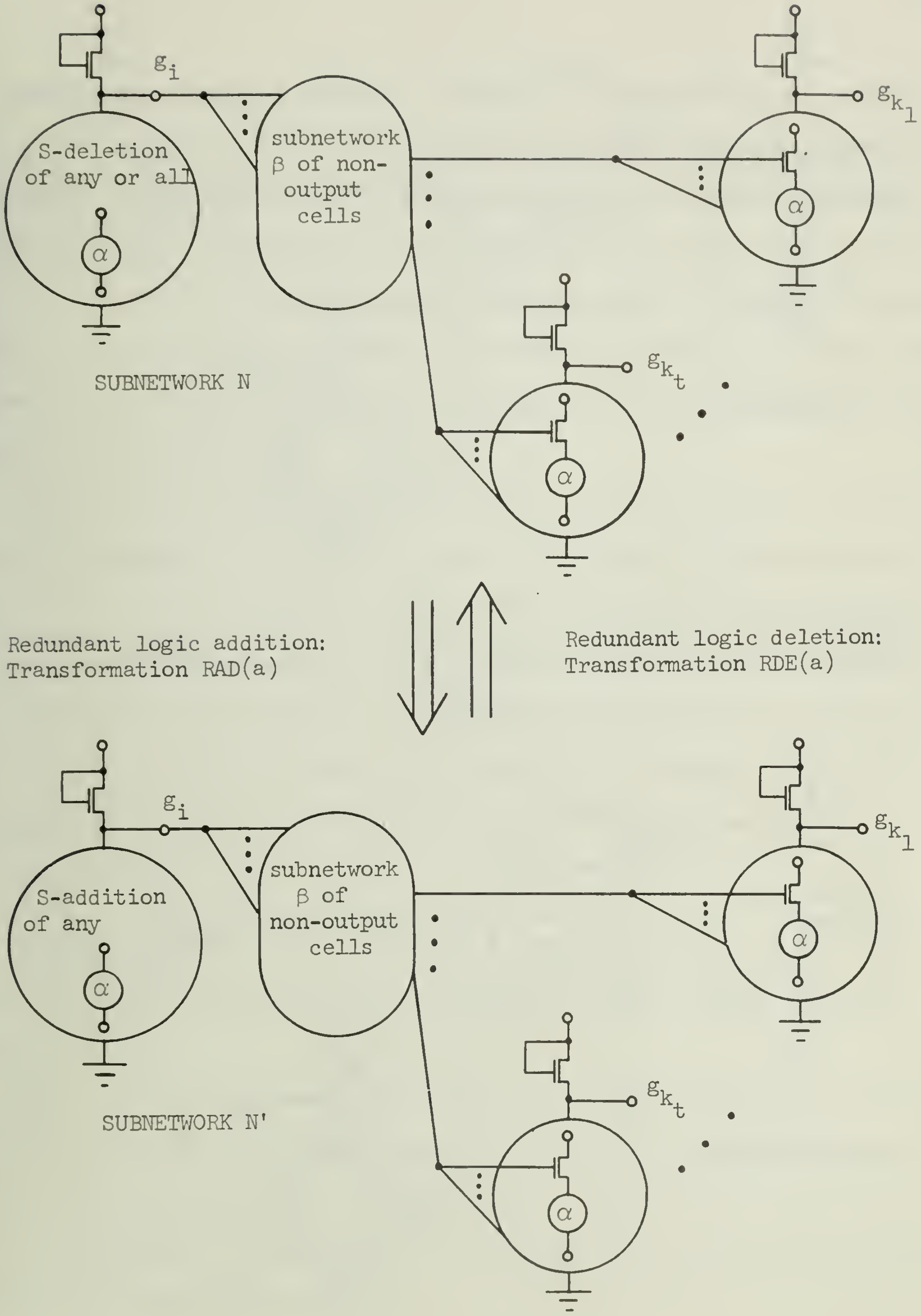

g_i
S-deletion of any or all
α
subnetwork β of non-output cells
g_{k_1}
α
g_{k_t}
α
SUBNETWORK N
Redundant logic addition: Transformation RAD(a)
Redundant logic deletion: Transformation RDE(a)
g_i
S-addition of any
α
subnetwork β of non-output cells
g_{k_1}
α
g_{k_t}
α
SUBNETWORK N'

Fig. 5.4.1 First pair of redundant logic addition/deletion transformations: RAD(a) and RDE(a).

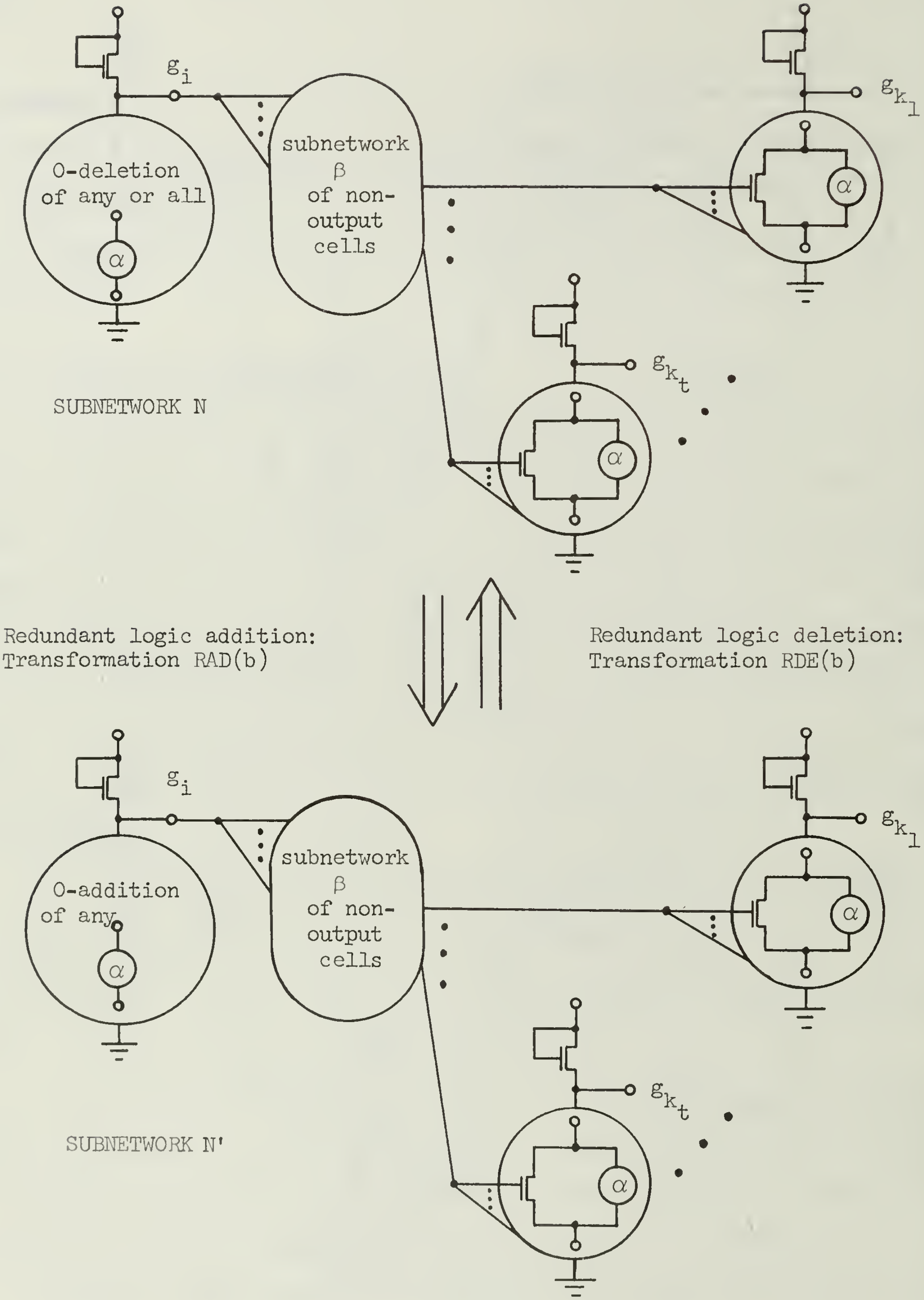

g_i
O-deletion of any or all
α
subnetwork β of non-output cells
g_{k_1}
α
g_{k_t}
α
SUBNETWORK N
Redundant logic addition: Transformation RAD(b)
Redundant logic deletion: Transformation RDE(b)
g_i
O-addition of any
α
subnetwork β of non-output cells
g_{k_1}
α
g_{k_t}
α
SUBNETWORK N'

Fig. 5.4.2 Second pair of redundant logic addition/deletion transformations: RAD(b) and RDE(b).

Transformation RAD(a) (RAD(b)) Consider an FET subnetwork of configuration α which appears in the drivers of one or more cells in an MOS network, N. Consider also a non-output cell g_i in N such that neither g_i nor its successors control any FET in configuration α. If, along every path from g_i to an output cell of N, there exists a non-output cell for which every FET controlled by its output is in series (parallel) with an FET subnetwork of configuration α, then one or more FET subnetworks of configuration α can be S-added (O-added) anywhere in the driver of g_i.

Transformation RDE(a) (RDE(b)) Consider an FET subnetwork of configuration α which appears in the drivers of one or more cells in an MOS network, N'. Consider also a cell g_i in N' such that neither g_i nor its successors control any FET in configuration α. If, along every path from g_i to an output cell of N', there exists a non-output cell for which every FET controlled by its output is in series (parallel) with an FET subnetwork of configuration α, then one or more FET subnetworks of configuration α can be S-deleted (O-deleted) anywhere in the driver of g_i.

Theorem 5.4.1 The transformation of a subnetwork of a given network by either Transformation RAD(a), RAD(b), RDE(a), or RDE(b) does not alter the network's outputs.

Proof Consider Transformation RAD(a) first. Suppose the theorem statement is not true. In other words, assume there exists a network N for which the function of at least one output cell, g_j, is changed due to the S-addition of FET subnetworks of configuration α to the driver of a cell g_i satisfying the conditions of Transformation RAD(a).

Since the configuration of g_i is the only one changed by RAD(a), only the functions realized by g_i and its successors may change as a result of the transformation. Let g_ℓ be a successor of g_i for which every FET connected to g_ℓ's output is in series with an FET subnetwork of configuration α. Whether or not the function realized by g_ℓ is changed as a result of the transformation, it can be shown that it cannot contribute to any change of function in any of its immediate successors. Consider the following two possible cases for an input vector A of the network, noting that the transmission function of FET subnetwork α is invariant under the transformation since no input of α is g_i or a successor of g_i:

Case 1 The transmission function of α is 1 for A. Since Transformation RAD(a) only replaces short circuits in the driver of g_i by FET subnetworks of configuration α and since such FET subnetworks effectively become the short circuits they replace for input vectors A for which α is conducting, the transmission function of the entire driver of g_i for each such A must be invariant under the transformation. Thus, for A, the outputs of g_i and its successors, including g_ℓ and its successors, are unchanged by the transformation.

Case 2 The transmission function of α is 0 for A. Since every FET, F, connected to g_ℓ's output is in series with an FET subnetwork of configuration α, for such an input vector A, no conducting path through the driver of an immediate successor of g_ℓ can include F. Thus, for A, the functions of the immediate successors of g_ℓ are independent of any change in the function realized by g_ℓ which may result from the transformation.

Now consider the output cell g_j whose realized function is changed by RAD(a). Since the configuration of cell g_j is not altered by the transformation, the only other possible cause of a change in function is an input of g_j whose realized function is also changed by RAD(a). Since the external variables obviously can not be affected by RAD(a), this means at least one cell, g_p, for which at least one FET connected to its output is not in series with a subnetwork α, is an immediate predecessor of g_j whose realized function is affected by RAD(a). The same argument can be made for the existence of a similar immediate predecessor, g_q, of g_p and so forth. Due to the assumption of a loop-free, finite network, a 'chain' of cells created in this manner, realizing functions altered as a result of RAD(a), must eventually be found to originate with g_i (since only g_i was directly affected by RAD(a), only the functions realized by its successors may be changed).

Thus, this chain of cells represents a path extending from g_i to g_j along which no cell g_ℓ, $\ell \neq j$, exists for which every FET connected to its output is in series with an FET subnetwork of configuration α. The existence of such a path, however, contradicts the assumption that network N satisfies the conditions of Transformation RAD(a) for a cell g_i.

Therefore, no network exists whose network outputs are altered by the proper use of Transformation RAD(a), and, consequently, the theorem statement concerning RAD(a) is true.

Similar arguments can prove the theorem for the other three transformations: RAD(b), RDE(a), and RDE(b).

Q.E.D.

It may be noted that the conditions required for the preceding two pairs of MOS network transformations have a rough counterpart in the generalized triangular connection configuration for networks of NOR gates (among others). (This configuration is a special case of Property 3 given in [LNM 74].) This configuration is illustrated in Fig. 5.4.3. NOR gates in subnetwork S have output connections only to each other or to gates $k_1,...,k_s$. NOR gate j has an output connection to each of $k_1,...,k_s$. In such a case, connections from j to the NOR gates of subnetwork S may be added or removed as desired without affecting the output functions of $k_1,...,k_s$ or their successors. (A slightly less generalized triangular connection configuration for NOR gates is shown in [BILM 69].)

Transformations RAD(a) and RAD(b) may be used to change the configuration of a cell in anticipation of further network transformation (e.g., by Transformation COM). An obvious application of RDE(a) or RDE(b) is the reduction of the number of FET's in a network. Fig. 5.4.4 and Fig. 5.4.5 show two examples of such usage of RDE(b). The two original networks used in these figures are taken from [Shi 72], p. 116, which discusses a computer program implementation and subsequent results of the non-level-restricted MOS network synthesis algorithm proposed in [Liu 72].

In the network of Fig. 5.4.4(a) the connection from cell g_1 to g_2 is actually unnecessary. The fact can be recognized and the FET in g_2 connected to the output of g_1 deleted through the use of RDE(b) as follows: (1) The driver of cell g_3 in Fig. 5.4.4(a) can be transformed into the new configuration in Fig. 5.4.4(b) by simple 'logic defactorization' (discussed in Section 5.5). (2) Considering an FET connected to the output of g_1 to be the necessary FET configuration α, it can be seen that α occurs in the driver of g_2 and in parallel with the only FET connected to the output of g_2. The conditions for

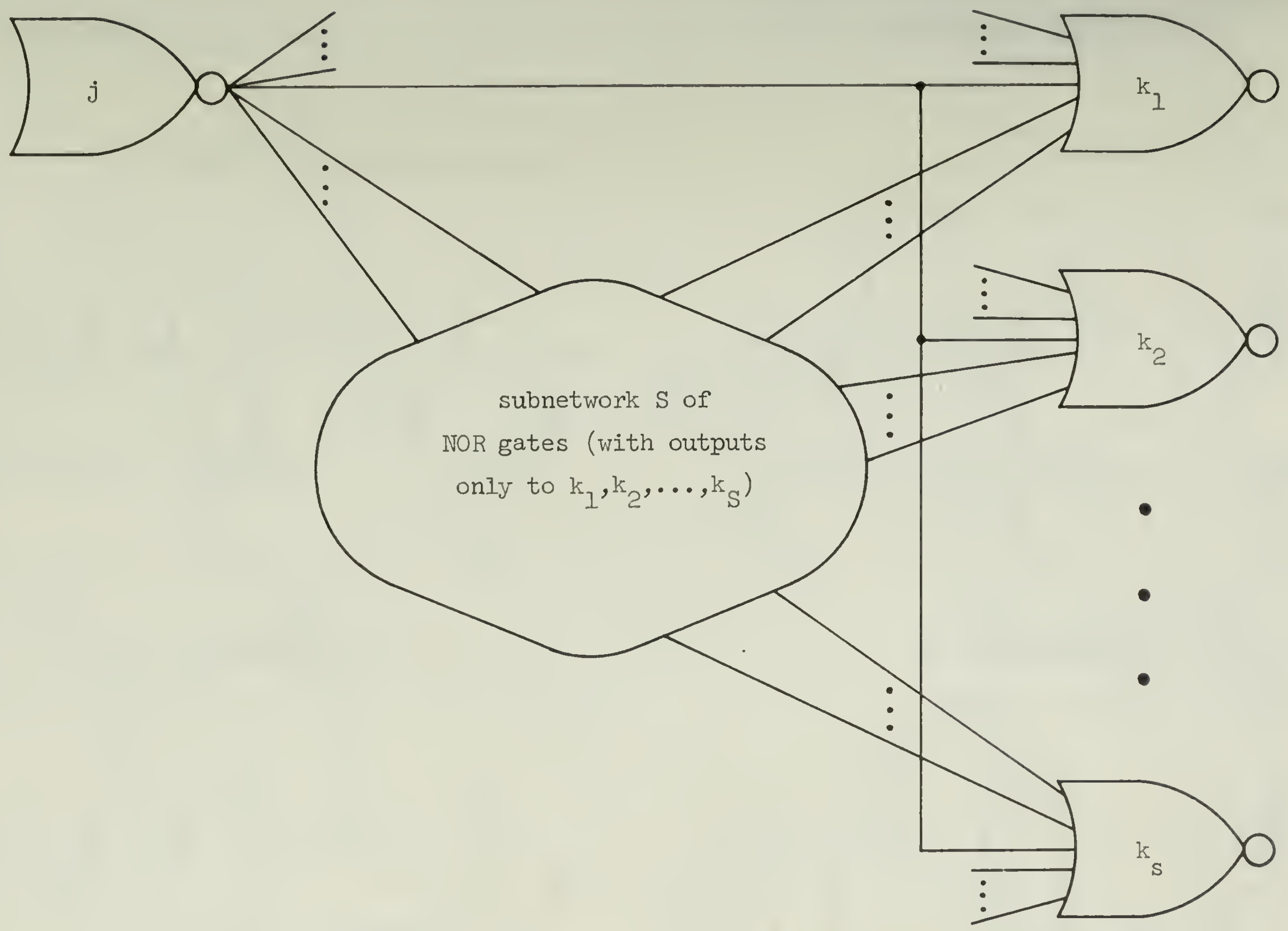
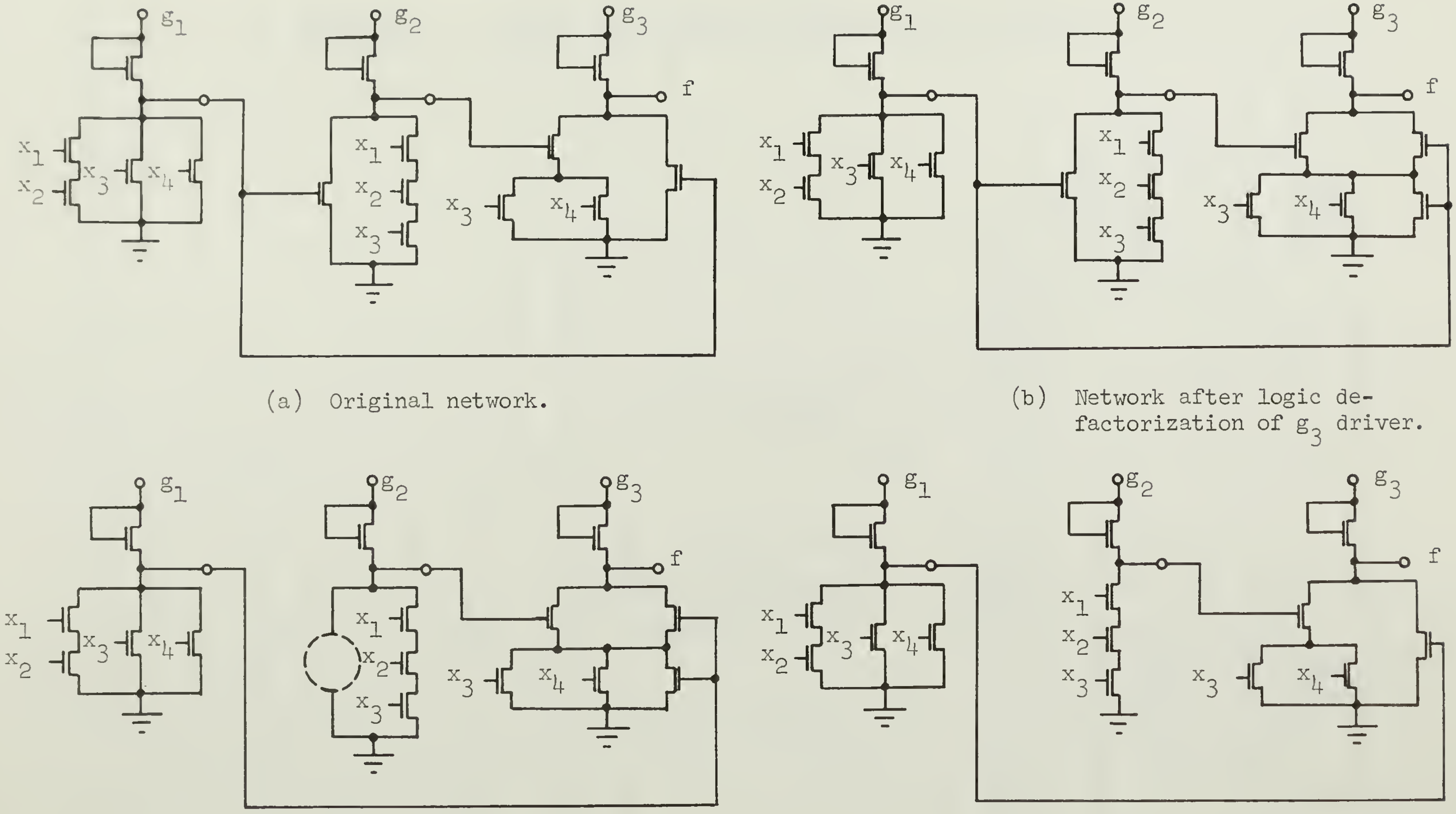


Fig. 5.4.3 Generalized triangular connection for NOR gates.



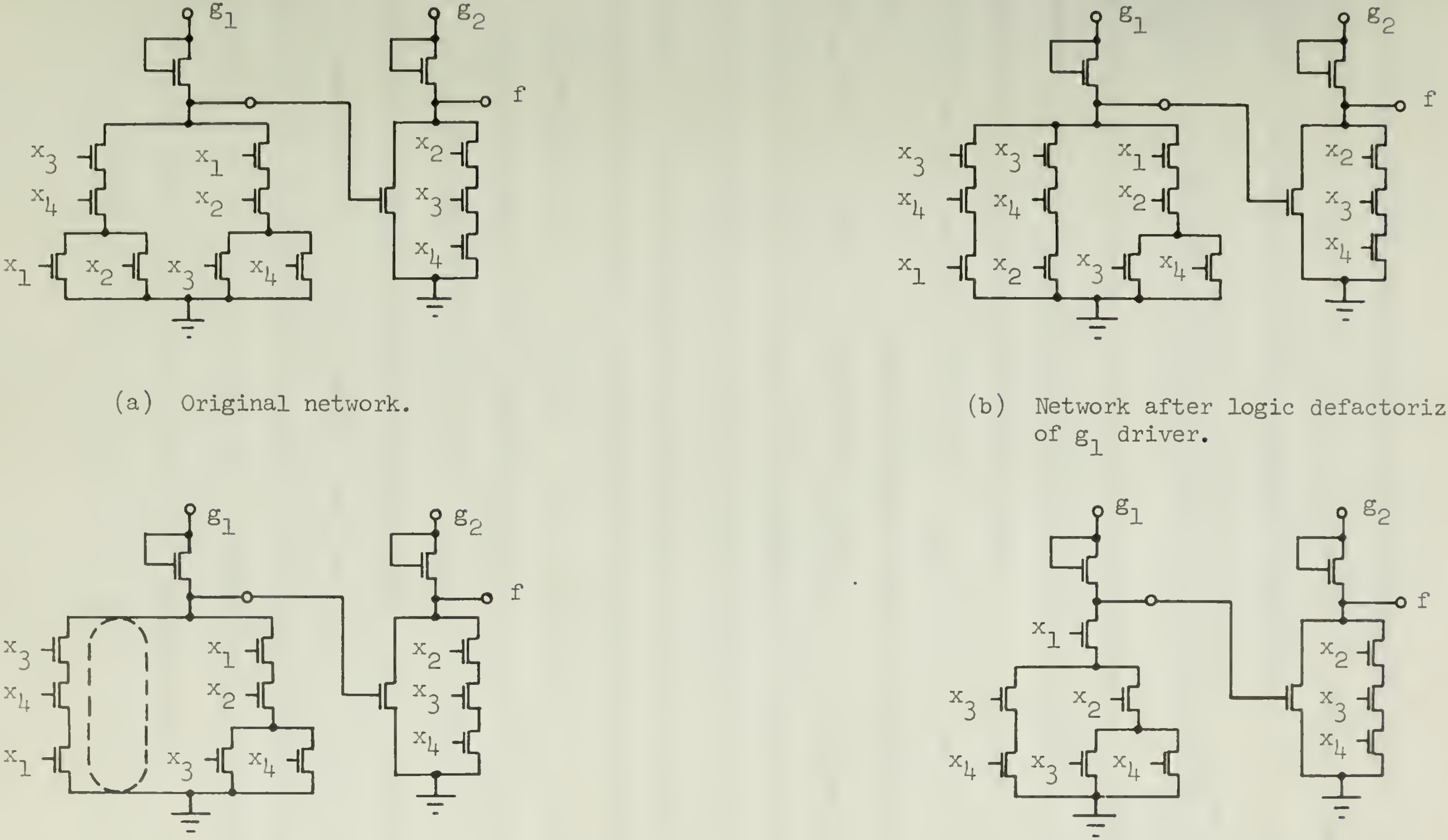
(a) Original network.

(b) Network after logic de-factorization of g_3 driver.

(c) Network after transformation by RDE(b).

(d) Final network after logic factorization of g_3 driver.

Fig. 5.4.4 First example of redundant FET removal through Transformation RDE(b).



(a) Original network.

(b) Network after logic defactorization of g_1 driver.

(c) Final network after transformation by RDE(b).

(d) Subsequent improvement by logic factorization.

Fig. 5.4.5 Second example of redundant FET removal through Transformation RDE(b).

the application of Transformation RDE(b) being thus satisfied, subnetwork α can be 0-deleted from the driver of g_2 to obtain the result in Fig. 5.4.4(c). (3) This result can be transformed by simple 'logic factorization' (also discussed in Section 5.5) of the driver of g_3 to achieve the final result in Fig. 5.4.4(d).

While deriving the final network in this example formally involves the additional use of intra-cell transformations (logic factorization/defactorization), an experienced user could skip these steps by recognizing that the FET in g_3 connected to the output of g_2 is 'effectively' in parallel with a subnetwork of configuration α in the original network.

A second example of the capability of RDE(b) is illustrated in Fig. 5.4.5. In this case, the subnetwork configuration α consists of FET's connected to external variables. The example proceeds as follows: (1) The driver of cell g_1 in Fig. 5.4.5(a) is first reconfigured by logic defactorization (resulting in Fig. 5.4.5(b)). (2) The serial configuration of three FET with connections from x_2, x_3, and x_4 is selected to be α. In Fig. 5.4.5(b) α appears both in the driver of g_1 and in g_2 in parallel with the only FET connected to g_1. Actually, the three FET's of α appear in a different serial order in g_2, but if this configuration can not be recognized as being equivalent to α (e.g., when transformations are being performed by machine), a basic intracell transformation discussed in Section 5.5 is capable of permuting the series of three FET's into the exact configuration α. (3) With this choice of α, Transformation RDE(b) is seen to be applicable, and the FET subnetwork of configuration α can can be 0-deleted from the driver of cell g_1 to obtain the result in Fig. (c).

For this example, in addition to the FET removed by RDE(b), a second FET can be saved by logic factorization of the driver of cell g_1 (the two FET's connected to x_1 can be combined - see Fig. 5.4.5(d)).

5.5 Logic Factorization/Defactorization and Other Basic Intracell Transformations

This class of widely known and basic transformations of FET networks is included just for the logical completeness of the proposed set of transformations based on configuration considerations.

They are often necessary in situations where the transformations described in previous sections (5.1 through 5.4) are applied rigorously (as in computer implementations). In hand applications, these intracell transformations often need not be actually carried out, the recognition of the equivalence of different FET configurations sufficing to enable the evaluation of the applicability of more sophisticated transformations (e.g., not all of the detailed steps in the two examples at the end of the preceding section need be carried out in practice).

The transformations are illustrated in Fig. 5.5.1 and Fig. 5.5.2. α, β, and γ are three possibly different FET subnetwork configurations having the respective transmission functions f_α, f_β, and f_γ. The two pairs of logic factorization/defactorization transformations (Fig. 5.5.1) can simply be justified by the logical equivalence of the Boolean forms $(f_\alpha \vee f_\beta)f_\gamma$ and $f_\alpha f_\gamma \vee f_\beta f_\gamma$ and of $f_\alpha f_\beta \vee f_\gamma$ and $(f_\alpha \vee f_\gamma)(f_\beta \vee f_\gamma)$, respectively. The remaining basic intracell transformations (Fig. 5.5.2) are even more obvious.

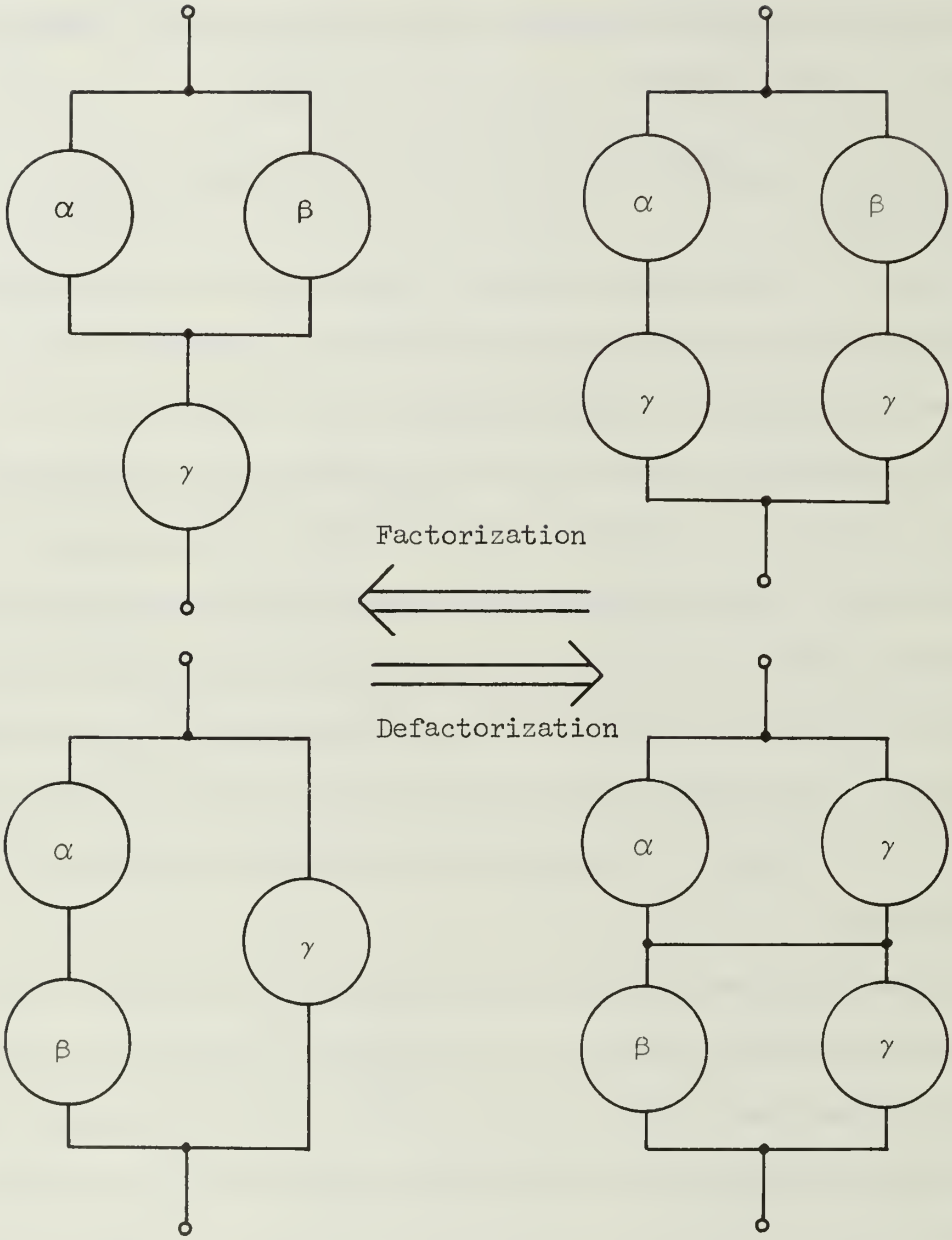


Fig. 5.5.1 Logic factorization/defactorization (intracell) transformations.

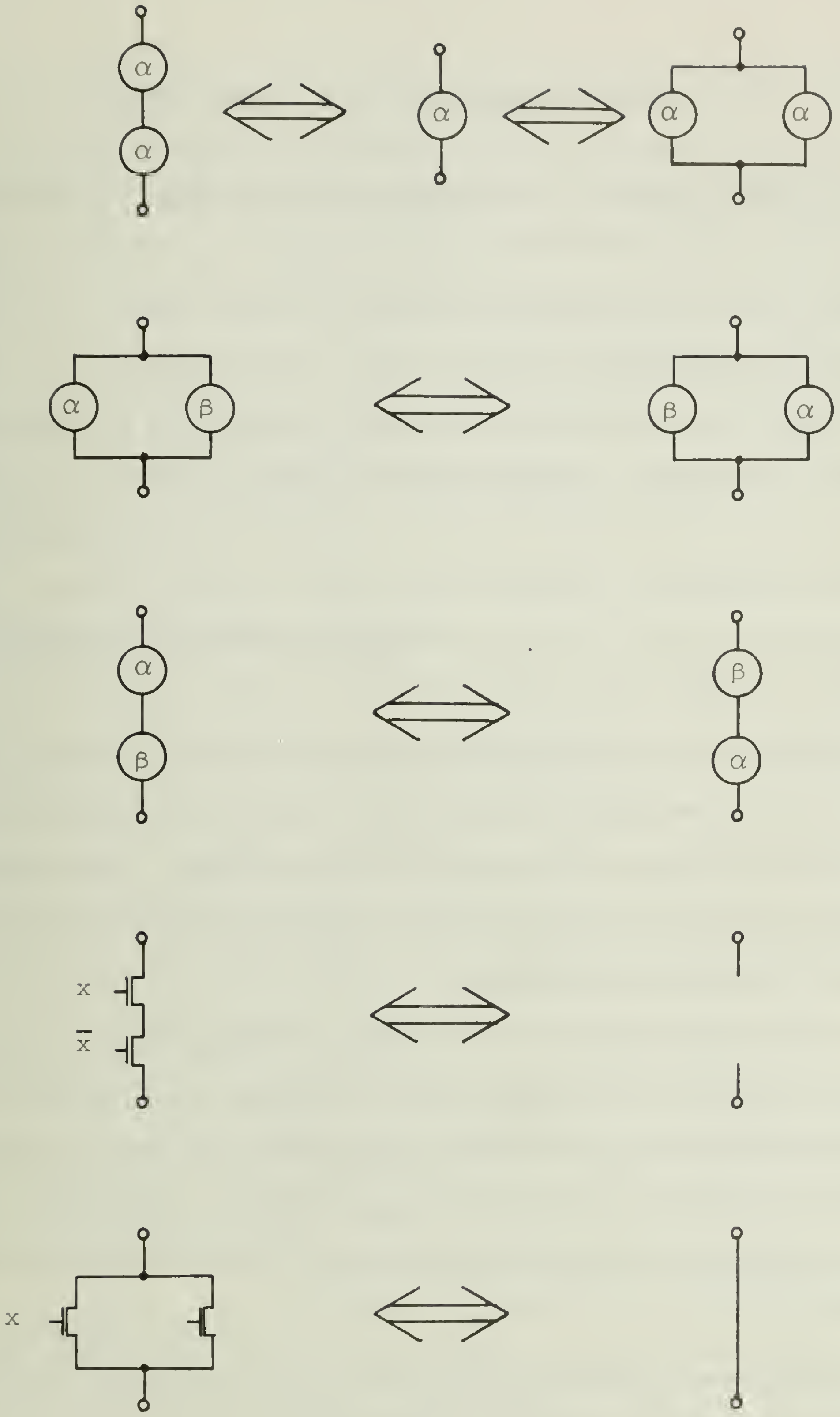


Fig. 5.5.2 Other basic intracell transformations.

6. TRANSDUCTION APPROACH TO MOS NETWORK DESIGN

In Section 4, methods to synthesize networks, usually G-minimal or G-minimal under certain restrictions, were discussed. A different approach to the design of networks of MOS cells is proposed here: Given a desired set of (completely or incompletely specified output functions, $f_1,\ldots,f_m$, and an existing network realizing these functions (it need not be G-minimal), the network is transformed by a procedure which attempts to reduce the numbers of cells and intercell connections and/or to 'simplify' the functions realized by the cells (allowing a reduction of the number of FET's employed). Such procedures will be called transduction (transformation and reduction) procedures.

The transduction procedures to be proposed here are, in a sense, more powerful than the transformations described in Section 5 which are based on the consideration of the cell and network configurations. Transduction procedures are based on a consideration of the specific function realized by or required of each cell of the network.

The transduction approach to MOS design was suggested by the success of the research group — of which the author was a member — led by Prof. Muroga of the Department of Computer Science, University of Illinois, in the use of transduction methods to obtain 'near-optimal' networks of NOR gates within a reasonable expenditure of computational effort. [Cul 75][CLK 74][KC 76][KM 76][KLM tbp][KLCM 75][Lai 75][LC 74][LC 75][LK 75] Although there are similarities in the transduction approaches to the design problem for negative gate networks and that for NOR gate networks, the two approaches are also different in many respects — stemming mainly from the fact that the actual function realized by a negative gate is not fixed as in the NOR gate case. The relationship of these two design problems will be further discussed later as different aspects of the proposed transduction approach to MOS network design are developed.

Section 6.1 will treat the theoretical development of transduction procedures for MOS networks, and Section 6.2 will discuss actual computer program implementations of these transduction procedures, as well as examples of results obtained. The usefulness of the transduction approach will be discussed in Section 6.3.

6.1 MOS Network Transduction Procedures

Consider a negative gate or MOS cell network with inputs $x_1,\dots,x_n$, (completely or incompletely specified) output function f, and gates $g_1,\dots,g_R$ realizing functions $u_1,\dots,u_R = f$ (with respect to $x_1,\dots,x_n$). For each negative gate, g_i, realizing function u_i, u_i must be a negative function of its inputs. In other words, according to Theorem 2.4, for every 0-1 pair of u_i, one of the inputs of g_i must provide a 1-0 cover.[†] As long as this condition is met, g_i can remain a negative gate realizing function u_i — even if those values of its inputs not involved in the 1-0 covers are changed and even if new inputs are added, old inputs are removed, and/or new inputs are substituted for old. Transduction procedures are based on the fact that if 1-0 covers provided by inputs of a gate are 'guaranteed' for all 0-1 pairs of the gate's

[†] Compare this with the case for NOR gates: (1) for each specified '1' of u_i (suppose u_i is the function of NOR gate g_i), the function of every input of g_i must be a '0'; (2) for each specified '0' of u_i, at least one of the inputs must provide a '1' cover.

output, then all of the other values of all of its inputs, i.e., those not involved in any such 1-0 cover, may be considered to be *'s with respect to that particular gate.

The following definitions will allow a presentation of systematic procedures for network transduction.

Definition 6.1.1 Let an augmented truth table for a gate g_j in a network N with n external variables and R negative gates be a truth table with 2^n rows, one for each input vector $A \in V_n$, and $R + p_j$ columns, one for each gate, g_1, $\ldots, g_R$, plus p_j supplementary columns, $s_1, \ldots, s_{p_j}$. An entry in row A and column g_j of the augmented truth table is $g_j(A)$, which may be 1, 0 or * (unspecified). Each supplementary column s_i corresponds to a distinct pair consisting of a 0 entry and a 1 entry in column g_j (i.e., a 0-1 pair of function u_j). A supplementary column has only two non-blank entries duplicating the pair of 0 and 1 entries to which it corresponds.

Example 6.1.1 An example of an augmented truth table for a gate g_3 in a network consisting of three negative gates (see Fig. 6.1.1) is shown in Fig. 6.1.2. Column g_3 has three 1 entries and five 0 entries and therefore requires the 15 supplementary columns shown. Assume that external variables x_1, x_3 and negative gates g_1, g_2 are inputs of gate g_3. Additional rows for illustrational purposes have been added below the supplementary columns. In this case, inputs which have corresponding 1-0 covers for each 0-1 pair of g_3 are noted. For example, the 0-1 pair of g_3 represented by supplementary column s_7 has corresponding 1-0 covers in both x_3 and g_2.

Definition 6.1.2 Let an augmented truth table be constructed for a gate g_i with inputs $x_{k_1}, \ldots, x_{k_r}$, $g_{\ell_1}, \ldots, g_{\ell_s}$, in a network N. An assignment

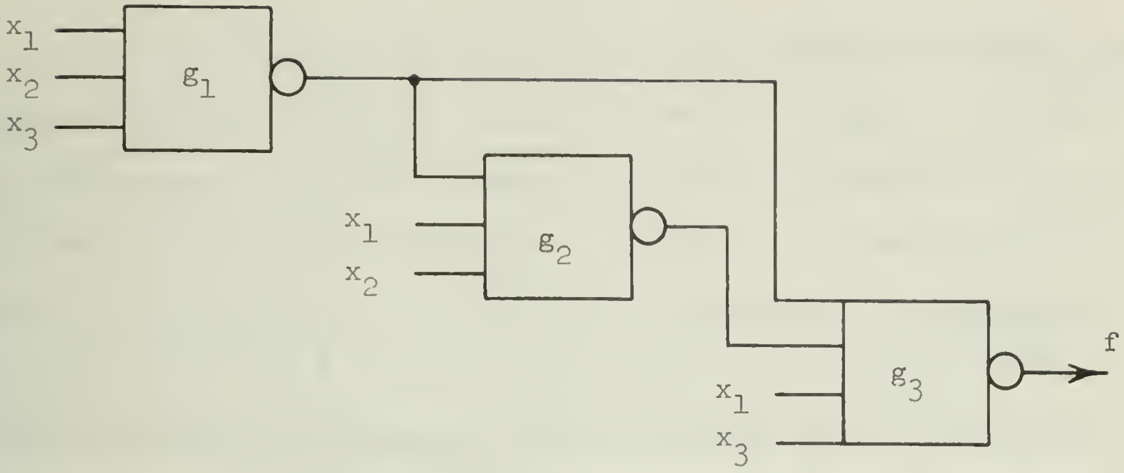


Fig. 6.1.1 Network of Example 6.1.1 realizing $f = x_1x_2 \vee x_2\bar{x}_3$.

x_1	x_2	x_3	g_1	g_2	g_3	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
0	0	0	1	0	0	0					0					0				
0	0	1	0	1	0		0					0					0			
0	1	0	0	1	1	1	1	1	1	1										
0	1	1	0	1	0			0					0					0		
1	0	0	0	1	0				0					0					0	
1	0	1	0	1	0					0					0					0
1	1	0	0	0	1						1	1	1	1	1					
1	1	1	0	0	1											1	1	1	1	1
existing 1-0 covers					x_1				×	×										
					x_3		×	×		×		×	×		×					
					g_1	×					×					×				
					g_2							×	×	×	×		×	×	×	×

Fig. 6.1.2 Augmented truth table for g_3 of Example 6.1.1.

of covers for column g_i with respect to the set of columns $x_{k_1},\ldots,x_{k_r}$, g_{ℓ_1}, $\ldots,g_{\ell_s}$, is a selection of one 1-0 cover in the set of columns for each 0-1 pair in a supplementary column of the truth table. The 0 entry and 1 entry of the table involved in each such 1-0 cover will be called, respectively, a necessary 0 and a necessary 1 for column g_i. If exactly one 1-0 cover exists among columns $x_{k_1},\ldots,x_{k_r}$, $g_{\ell_1},\ldots,g_{\ell_s}$ for a particular supplementary column, this 1-0 cover will be called an essential 1-0 cover with respect to the 0-1 pair and the set of columns $x_{k_1},\ldots,x_{k_r}$, $g_{\ell_1},\ldots,g_{\ell_s}$. Furthermore, the external variable or gate corresponding to the column containing the essential 1-0 cover is called an essential input of gate g_i with respect to the set of inputs $x_{k_1},\ldots,x_{k_r}$, $g_{\ell_1},\ldots,g_{\ell_s}$.

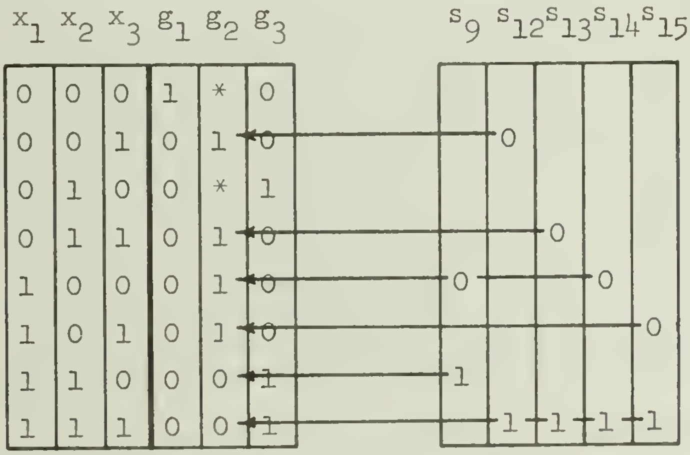
Note that in the worst case, when half of the values in column g_i are 0's and half are 1's, there are $2^{n-1} \times 2^{n-1} = 2^{2n-2}$ 0-1 pairs which require the assignment of 1-0 covers — a number which grows rapidly with n.[†] Fortunately, as can be seen in the computer program implementation of the transduction procedures (described in Section 6.2.1), by properly organizing the assignment of covers, not all covers need be explicitly assigned (some are implicitly assigned by the assignment of others).

Example 6.1.2 Fig. 6.1.3 again shows the augmented truth table of Example 6.1.1 for a gate g_3 assumed to have inputs x_1, x_3, g_1, and g_2. An assignment of covers for column g_3 has been made as indicated in the rows below the supplementary columns. As seen in Fig. 6.1.2, supplementary columns

[†] In the case of NOR gates, no more than 2^{n-1} 'covers' need be assigned for any single gate. Also, a 'cover' in the case of NOR gates is a single value, while a cover in the case of negative gates is a pair of values.

x_1	x_2	x_3	g_1	g_2	g_3	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
0	0	0	1	0	0	0					0					0				
0	0	1	0	1	0		0					0					0			
0	1	0	0	1	1	1	1	1	1	1										
0	1	1	0	1	0			0					0					0		
1	0	0	0	1	0				0					0					0	
1	0	1	0	1	0					0					0					0
1	1	0	0	0	1						1	1	1	1	1					
1	1	1	0	0	1											1	1	1	1	1
assigned 1-0 covers					x_1				×											
					x_3		×	×		×		×	×		×					
					g_1	×					×					×				
					g_2									×			×	×	×	×

Fig. 6.1.3 Augmented truth table with covers assigned for g_3 of Examples 6.1.1 and 6.1.2.



x_1	x_2	x_3	g_1	g_2	g_3	s_9	s_{12}	s_{13}	s_{14}	s_{15}
0	0	0	1	*	0					
0	0	1	0	1	0		0			
0	1	0	0	*	1					
0	1	1	0	1	0			0		
1	0	0	0	1	0	0			0	
1	0	1	0	1	0					0
1	1	0	0	0	1	1				
1	1	1	0	0	1		1	1	1	1

Fig. 6.1.4 Determining necessary 1's and 0's in g_2 of Examples 6.1.1 and 6.1.2.

s_1, s_2, s_3, s_4, s_6, s_9, s_{11}, s_{12}, s_{13}, s_{14}, and s_{15} have corresponding essential 1-0 covers. 1-0 covers assigned in column g_2 for 0-1 pairs of column g_3 create the four necessary 1's and two necessary 0's in column g_2 as shown in Fig. 6.1.4.

After all of the 1-0 covers have been assigned for every column representing an immediate successor of a gate g_i, all of the 1-0 covers required to be provided by g_i are known, and hence all of the necessary 1's and necessary 0's of g_i are known. Every other entry of column g_i which is not a necessary 1 or necessary 0 can then be changed to a 'don't care' since their exact values are not important at this stage. The only exception is when g_i is an output gate of the network which must realize some output function f_j. In such a case, all specified 0's and 1's of function f_j must also be considered necessary 0's and 1's of column g_i since any change of these values would obviously cause an erroneous output.

Example 6.1.3 Since the only immediate successor of g_2 in Examples 6.1.1 and 6.1.2 is g_3, the only necessary 0's and 1's (shown in Fig. 6.1.4) of column g_2 are the necessary 0's and 1's resulting from 1-0 covers assigned for g_3. Hence the two remaining values, in the first and third rows of column g_2, may be changed to 'don't cares.' An assignment of 1-0 covers can now be made for column g_2, with respect to columns x_1,x_2, and g_1 (corresponding to inputs of gate g_2 in the network), ignoring the '*' entires of the column. The augmented truth table for g_2 and one possible assignment of 1-0 covers is shown in Fig. 6.1.5. In this case, g_1 provides no 1-0 covers for g_2, and it is thus seen that the connection from g_1 to g_2 can be removed. It is important to note that had the entry in row 1 and column g_2 not been recognized as being 'don't care.' g_1 would have been regarded as an essential input of g_2, e.g., due to the 0-1 pair in rows 1 and 2 and column g_2 (see Fig. 6.1.2) which would otherwise exist and for which only g_1 could provide a 1-0 cover. Now consider column g_1. Since the only necessary 1's and 0's in column g_1 are a result of providing three 1-0 covers for g_3, all of the other components, unnecessary 1's and 0's, can be set to 'don't cares' (see Fig. 6.1.6). The augmented truth table for g_1 (Fig. 6.1.7) shows that only x_2 is an essential input for g_1, and it is alone sufficient to cover every remaining 0-1 pair of g_1. Thus, the connections from x_1 and x_3 to g_1 can also be eliminated as redundant. At this point, the corresponding negative gate network is shown in Fig. 6.1.8 with its associated truth table.

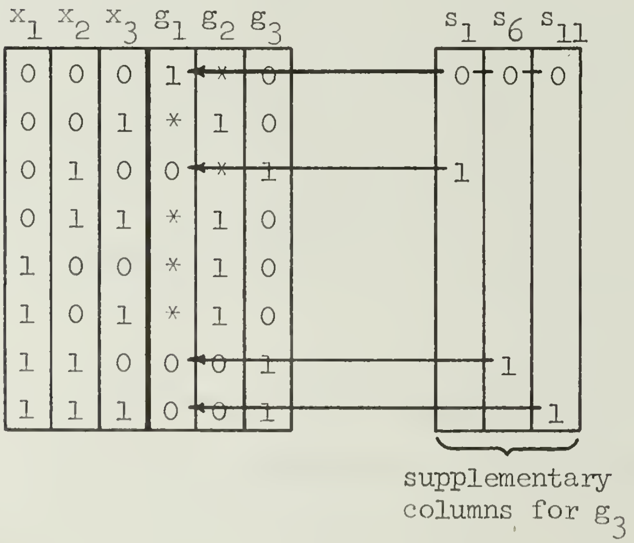
The preceding examples have demonstrated how a negative gate network whose individual gate outputs are known can be transformed, while maintaining required output functions, on the basis of Theorem 2.4 and the concept of 1-0 cover assignment. Note at this point that the transformed network is expressed only as an abstract negative gate network — a specification of connections among external variables and gates and a truth table giving the output function of each gate of the network. If an MOS implementation of this negative gate network is desired, additional computations are required, regardless of whether or not an MOS implementation of the original network was known. This will be discussed further later.

The type of transduction procedure exemplified above is referred to as a pruning procedure[†] since it transforms a network only by 'pruning' existing

[†] [CLK 74] and [LC 74] discuss pruning procedures for NOR gates.

	x_1	x_2	x_3	g_1	g_2	g_3	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
	0	0	0	1	*	0								
	0	0	1	0	1	0	1	1						
	0	1	0	0	*	1								
	0	1	1	0	1	0			1	1				
	1	0	0	0	1	0					1	1		
	1	0	1	0	1	0							1	1
	1	1	0	0	0	1	0		0		0		0	
	1	1	1	0	0	1		0		0		0		0
existing 1-0 covers						x_1	×	×	×	×				
						x_2	×	×			×	×	×	×
						g_1								
assigned 1-0 covers						x_1			×	×				
						x_2	×	×			×	×	×	×

Fig. 6.1.5 Augmented truth table with covers assigned for g_2 of Examples 6.1.1 - 6.1.3.

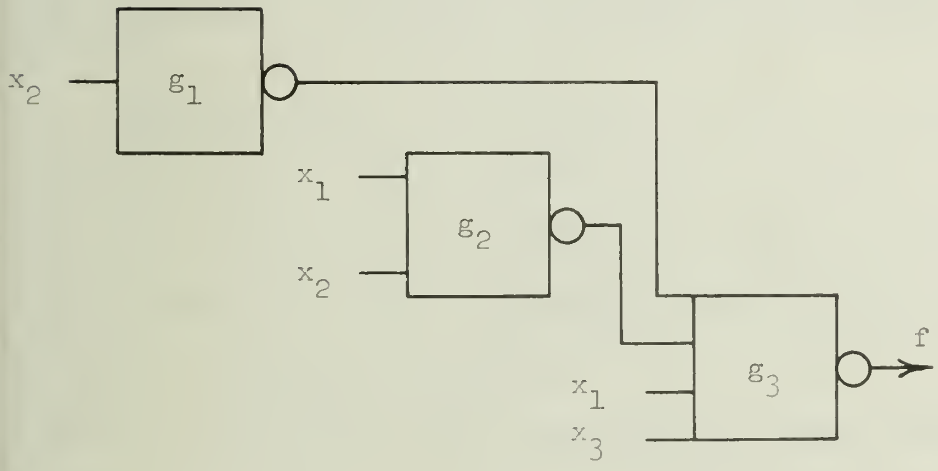


x_1	x_2	x_3	g_1	g_2	g_3	s_1	s_6	s_{11}
0	0	0	1	*	0	0	0	0
0	0	1	*	1	0			
0	1	0	0	*	1	1		
0	1	1	*	1	0			
1	0	0	*	1	0			
1	0	1	*	1	0			
1	1	0	0	0	1		1	
1	1	1	0	0	1			1

Fig. 6.1.6 Determining necessary 1's and 0's in g_1 of Examples 6.1.1 - 6.1.3.

		x_1	x_2	x_3	g_1	g_2	g_3	s_1	s_2	s_3
		0	0	0	1	*	0	1	1	1
		0	0	1	*	1	0			
		0	1	0	0	*	1	0		
		0	1	1	*	1	0			
		1	0	0	*	1	0			
		1	0	1	*	1	0			
		1	1	0	0	0	1		0	
		1	1	1	0	0	1			0
existing 1-0 covers	x_1								×	×
	x_2							×	×	×
	x_3									×
assigned 1-0 covers	x_2							×	×	×

Fig. 6.1.7 Augmented truth table with covers assigned for g_1 of Examples 6.1.1-6.1.3.



x_1	x_2	x_3	g_1	g_2	g_3
0	0	0	1	*	0
0	0	1	*	1	0
0	1	0	0	*	1
0	1	1	*	1	0
1	0	0	*	1	0
1	0	1	*	1	0
1	1	0	0	0	1
1	1	1	0	0	1

Fig. 6.1.8 Simplified network realizing $f = x_1x_2 \vee x_2\bar{x}_3$ and associated truth table after covers assigned for g_1 of Examples 6.1.1 - 6.1.3.

connections among external variables and gates. A formalization of one such pruning procedure will now be developed.

A 2^n-dimensional vector with 1, 0, and * ('don't care') entries will be used to express the set of all completely specified functions of n variables which can be obtained by specifying the * entries to every possible combination of 0's and 1's (e.g., (1*0*) denotes the set of completely specified, 2-variable functions {(1000), (1001), (1100), (1101)}). If A denotes such a vector, $A^{(d)}$ denotes the $d^{\underline{th}}$ component of the vector, i.e., a function value for the input vector $(a_1,\ldots,a_n)$ where $\sum_{i-1}^{n} 2^{n-i}a_i + 1 = d$. External variables, x_1, ..., x_n, are also represented by 2^n-dimensional vectors:

$$x_1 = (\underbrace{0 \quad . \quad . \quad . \quad 0}_{2^{n-1}\text{ entries}}\ \underbrace{1 \quad . \quad . \quad . \quad 1}_{2^{n-1}\text{ entries}})$$

$$x_2 = (\underbrace{0 \ . \ . \ . \ 0}_{2^{n-2}\text{ entries}}\ \underbrace{1 \ . \ . \ . \ 1}_{2^{n-2}\text{ entries}}\ \underbrace{0 \ . \ . \ . \ 0}_{2^{n-2}\text{ entries}}\ \underbrace{1 \ . \ . \ . \ 1}_{2^{n-2}\text{ entries}})$$

.

.

.

$$x_{n-1} = (0\ 0\ 1\ 1\ 0\ 0\ 1\ 1 \quad . \ . \ . \quad 0\ 0\ 1\ 1)$$

$$x_n = (0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \quad . \ . \ . \quad 0\ 1\ 0\ 1).$$

$x_i^{(d)}$ represents the $d^{\underline{th}}$ component of the vector expressing x_i.

With reference to a network N with n external variables, x_1, ..., x_n, which consists of R negative gates, $g_1,\ldots,g_R$, and realizes m (completely or incompletely specified) output functions, $f_1,\ldots,f_m$, the following are defined:

$v_1,\ldots,v_n$ denote 'input terminals' to which are attached external variables $x_1,\ldots,x_n$, respectively;

$v_{n+1},\ldots,v_{n+R}$ denote negative gates $g_1,\ldots,g_R$; respectively, of which $v_{n+1},\ldots,v_{n+m}$ realize network output functions $f_1,\ldots,f_m$, respectively;

$v_{n+R+1},\ldots,v_{n+R+m}$ denote 'output terminals' whose sole connections are from gates $v_{n+1},\ldots,v_{n+m}$, respectively;

c_{ij} denotes a connection from v_i to v_j (undefined for $i \geq n + R + 1$ or $j \leq n$);

$f(v_i)$ denotes the function realized at v_i.

The correspondence between the v_i and the gates, external variables, and output functions is illustrated in Fig. 6.1.9. For simplicity, input and output terminals will be omitted in subsequent illustrations.

Definition 6.1.3 If, by replacing the function realized by a particular external variable, gate, or connection with a function f, the function realized by every (other) gate of the network remains negative with respect to the functions of its inputs and all the network outputs remain as specified, then f is said to be a permissible function[†] for that external variable, gate, or connection, respectively. Let $(G(v_i)$ and $G(c_{ij})$ denote a set of permissible functions (not necessarily inclusive of all possible permissible functions) for gate or external variable v_i and connection c_{ij}, respectively.

[†] Permissible functions and CSPF's were first introduced for NOR gates in [KM 76]. These corresponding definitions for negative gates take into account the fact that a negative gate can realize any negative function.

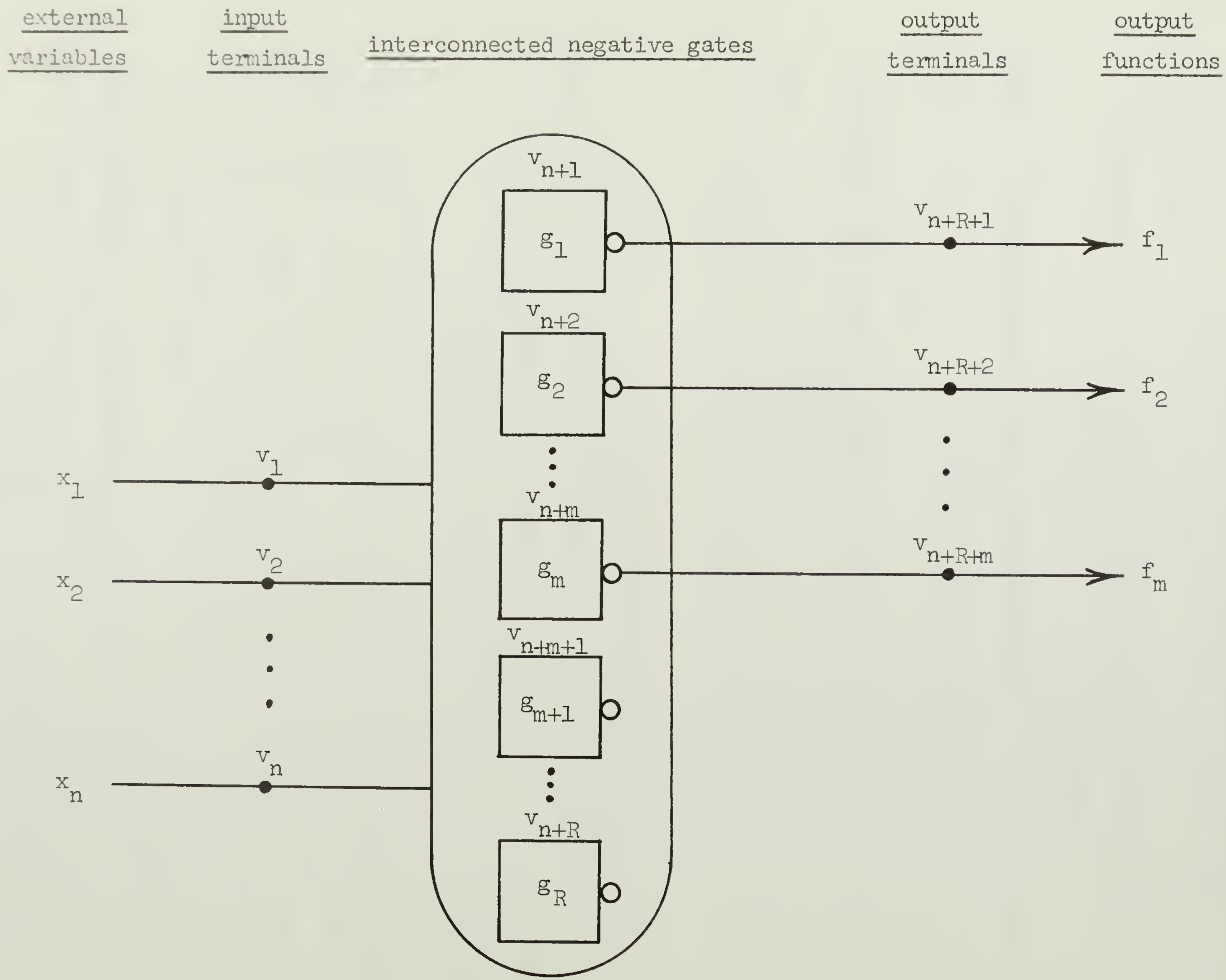


Fig. 6.1.9 Representation of networks to be treated by transduction procedures.

Definition 6.1.4 If a set of permissible functions is chosen for each gate, external variable, and connection of a network such that the following condition is satisfied for each gate g_i, then these sets are called compatible sets of permissible functions (CSPF's):[†]

> If any function f'_k is selected from the set of permissible functions of each immediate predecessor, v_k, or g_i or the set of permissible functions of its connection c_{ki}, then there exists a function f'_i in the set of permissible functions for g_i such that f'_i is negative with respect to the choice of f'_k.

A compatible set of permissible functions for a connection c_{ij} is denoted $G_c(c_{ij})$.

The CSPF's dealt with here will also be expressed in vector notation.

Three preference orderings of network elements (i.e., external variables and/or gates) will be defined in a general sense (the user will have some latitude in selecting the specific orderings) for use in the pruning procedure. These orderings are defined by three functions, σ_1, σ_2, and σ_3, which assume lower values for more preferred elements. For example, $\sigma_1(v_i) < \sigma_1(v_j)$ indicates that v_i is more preferred than v_j in preference ordering σ_1.

Preference ordering σ_1 is an ordering of gates for selection for pruning. This ordering must satisfy $\sigma_1(v_i) < \sigma_1(v_j)$ if $v_i \in S(v_j)$.

Preference ordering σ_2 is an ordering of gates and external variables according to their desirability for disconnection as inputs to other gates. During pruning, if a choice occurs of inputs to be disconnected from a certain gate v_j, the gate or external variable input, v_i, with lowest value $\sigma_2(v_i)$ among them is selected for disconnection (i.e., c_{ij} is removed from the network).

[†] See footnote on preceding page.

Preference ordering σ_3 is an ordering of gates and external variables according to their desirability for providing 1-0 covers. Generally, external variables are considered more preferable for covering since the 0's and 1's involved in such covers do not create 0-1 pairs which, subsequently, must themselves have assigned 1-0 covers (as is the case when a gate is assigned a 1-0 cover).

Algorithm 6.1.1 Procedure to 'prune' redundant connections from a negative gate network (PP).

A network N of n external variables, m output functions, and R negative gates will be transformed into a network N' having the same external variables and output functions. The number of connections among gates and external variables in N' will be less than or equal to that in N.

Step 1 Select a specific ordering σ_1 of gates and specific orderings σ_2 and σ_3 of gates and external variables (input terminals).

Step 2 Set $G_c(c_{i,i+R}) = f_{i-n}$, $i = (n + 1),\ldots,(n + m)$.

Step 3 Initially, let I be the set of all gates in N.

Step 4 Select $v_i \in I$ such that $\sigma_1(v_i) < \sigma_1(v)$ for every $v \in I$, $v \neq v_i$.

Step 5 Compute a CSPF for v_i as follows:

$$G_c(v_i) = \bigcap_{v_j \in IS(v_i)} G_c(c_{ij}).^{\dagger}$$

Step 6 Select an irredundant input set for v_i with respect to $G_c(v_i)$ as follows (this step performs the actual pruning):

[†] This calculation can be expedited by the use of the vector representations of the sets involved. It can be shown that: $G_c^{(d)}(v_i) = 1$ if and only if at least one $G_c^{(d)}(c_{ij})$ is a 1; $G_c^{(d)}(v_i) = 0$ if and only if at least one $G_c^{(d)}(c_{ij})$ is a 0; and $G_c^{(d)}(v_i) = *$ if and only if every $G_c^{(d)}(c_{ij})$ is a * (no other cases will occur).

(a) Seek a $v_j \in IP(v_i)$ such that:

(i) there exist no indices, d, e, satisfying: $G_c^{(d)}(v_i) = f^{(e)}(v_j) = 0$; $G_c^{(e)}(v_i) = f^{(d)}(v_j) = 1$; and either $f^{(d)}(v) = 0$ or $f^{(e)}(v) = 1$ for every $v \in IP(v_i)$, $v \neq v_j$. (This condition can be illustrated as follows:

$$\begin{array}{rl} & \qquad\quad \underline{d} \qquad\quad \underline{e} \\ G_c(v_i): & (\text{———}0\text{———}1\text{———}) \\ f(v_j): & (\text{———}1\text{———}0\text{———}) \\ IP(v_i) - \{v_j\}: & \left\{ \begin{array}{l} (\text{———}0\text{——————————}) \\ \qquad\qquad \text{or} \\ (\text{——————————}1\text{———}). \end{array} \right. \end{array}$$

If such indices, d and e, did exist, $f^{(e)}(v_j)$ and $f^{(d)}(v_j)$ would be an essential 1-0 cover and, hence, v_j would be an essential input of v_i.); and

(ii) for every other $v_k \in IP(v_i)$ satisfying condition (i), $\sigma_2(v_j) < \sigma_2(v_k)$.

(b) If no such v_j exists, continue to Step 7. Otherwise, $IP(v_i) \Longleftarrow IP(v_i) - \{v_j\}$ and return to substep (a).

Step 7 Calculate CSPF's for remaining input connections to v_i (assign 1-0 covers for 0-1 pairs of $G_c\ v_i)$)). Compatible to v_i (assign 1-0 covers for 0-1 paris of $G_c(v_i)$). Compatible sets of permissible functions are determined for remaining connections to v_i by the following specification of values:

$G_c^{(d)}(c_{ji}) = 0$ if and only if $f^{(d)}(v_j) = 0$, $G_c^{(d)}(v_i) = 1$ and there exists an index e such that: $f^{(e)}(v_j) = 1$, $G_c^{(e)}(v_i) = 0$, and no v, $v \neq v_j$ satisfies $v \in IP(v_i)$, $\sigma_2(v) < \sigma_3(v_j)$, $f^{(d)}(v) = 0$, and $f^{(e)}(v) = 1$;

$G_c^{(d)}(c_{ji}) = 1$ if and only if $f^{(d)}(v_j) = 1$, $G_c^{(d)}(v_i) = 0$, and there exists an index e such that: $f^{(e)}(v_j) = 0$, $G_c^{(e)}(v_i) = 1$, and no v, $v \neq v_j$ satisfies $v \in IP(v_i)$, $\sigma_3(v) < \sigma_3(v_j)$, $f^{(d)}(v) = 1$ and $f^{(e)}(v) = 0$;

$G_c^{(d)}(c_{ji}) = *$ otherwise.

Step 8 $I \Longleftarrow I - \{v_i\}$. If I is not empty, return to Step 4. Otherwise, terminate the algorithm. The vectors $G_c(v_i)$, $i = (n + 1),\ldots,(n + R)$, representing CSPF's for gates of the network define the incompletely specified functions realized by the respective negative gates in the new network N', i.e., $f'(v_i) = G_c(v_i)$, $i = (n + 1),\ldots,(n + R)$.

Theorem 6.1.1 The network N' obtained by Algorithm 6.1.1 (PP) is a negative gate network realizing the same set of m output functions, $f_1,\ldots,f_m$, realized by the original negative gate network, N. Furthermore, the set of connections among gates and external variables in such a network N' is a subset of that of N.

Proof The only change in gate interconnections effected by the algorithm occurs in Step 6b. Since this consists only of the removal of a connection from one of a gate v_i's inputs (although this may happen for several different v_i), the set of connections in N' must be a subset of the set of connections in N.

Let $IP(v_i)$ be $\{v_{j_1},\ldots,v_{j_{s_i}},\ldots,v_{j_{t_i}}\}$ in N and $\{v_{j_1},\ldots,v_{j_{s_i}}\}$ in N'. If it can be shown that for every function $f'(v_i)$, $i = (n + 1),\ldots,(n + R)$, determined for N', every 0-1 pair of $f'(v_i)$ has a 1-0 cover among specified values of functions $f'(v_{j_1}),\ldots,f'(v_{j_{s_i}})$ where $IP(v_i) = \{v_{j_1},\ldots,v_{j_{s_i}}\}$, then, by Theorem 2.4, $f'(v_i)$ is negative with respect to the functions of v_i's immediate predecessors and thus can be realized by negative gate v_i.

For every gate v_i, the sets of specified 1- and 0-components of the vector expressing $f'(v_i)$ must be subsets, respectively, of the sets of specified 1- and 0-components of the vector expressing $f(v_i)$ for the following reasons. The specified 1- and 0-components of (the vector expressing) $f'(v_i)$ are those of (the vector expressing) $G_c(v_i)$ as determined by the algorithm. The specified components of $G_c(v_i)$ are determined by a combination of Steps 7 and 5 of the algorithm. First, $G_c(c_{ij})$'s,$v_j \in IS(v_i)$, are specified by Step 7. It is clear that $G_c^{(d)}(c_{ij})$ can only be assigned the value 1 or 0 if $f^{(d)}(v_i)$ has the same specified value. Then $G_c(v_i)$ is formed in Step 5 by taking the intersection of the sets $G_c(c_{ij})$'s (note that by the definition of ordering σ_1, it is clear that all $G_c(c_{ij})$. $v_j \in IS(v_i)$, are already known when Step 5 is reached for v_i). Thus, every specified 0- or 1-component of a $G_c(c_{ij})$ must be a specified 0- or 1-component, respectively, of $G_c(v_i)$ (any vector not specified in this manner could obviously not represent a function in the intersection of the $G_c(c_{ij})$), and all other components of $G_c(v_i)$ are *'s (all combinations of specifications for these *'s obviously result in vectors representative of functions in the intersection of the $G_c(c_{ij})$). Therefore, every specified 1- or 0-component of $f'(v_i)$ ($G_c(v_i)$) is a specified 1- or 0-component, respectively, of $f(v_i)$.

Since by Theorem 2.4, every 0-1 pair in $f(v_i)$ must have at least one 1-0 cover among $f(v_{j_1}),\ldots,f(v_{j_{t_i}})$, every 0-1 pair in $G_c(v_i)$ – the set of such 0-1 pairs being a subset of the set of 0-1 pairs in $f(v_i)$ – must also have at least one 1-0 cover among $f(v_{j_1}),\ldots,f(v_{j_{t_i}})$. It can easily be seen that Step 6 only removes an input connection c_{ji} of v_i if all the 1-0 covers provided by v_j

for 0-1 pairs of $G_c(v_i)$ are not essential, i.e., if they are duplicated by other existing inputs. Hence, every 0-1 pair in $G_c(v_i)$ must have at least one 1-0 cover among $f(v_{j_1}),\ldots,f(v_{j_{s_i}})$. In Step 7, it is clear that for every 0-1 pair consisting of components d and e of $G_c(v_i)$, there exists a unique connection c_{ji} such that components $G_c^{(d)}(c_{ji})$ and $G_c^{(e)}(c_{ji})$ of $G_c(c_{ji})$ are assigned the 1 and 0 values necessary to provide a corresponding 1-0 cover. Hence any set of functions obtained by selecting one permissible function among each set $G_c(c_{ji})$ for each remaining input connection, c_{ji}, of v_i, provides a 1-0 cover for every 0-1 pair of $G_c(v_i)$. In particular, since $G_c(v_{j_1}),\ldots,$ $G_c(v_{j_{s_i}})$ are subsets of the sets of permissible functions $G_c(c_{j_1,i}),\ldots,$ $G_c(c_{j_{s_i},i})$, every 0-1 pair among specified values of $f'(v_i)$ has a 1-0 cover among specified values of functions $f'(v_{j_1}),\ldots,f'(v_{j_{s_i}})$. Therefore, $f'(v_i)$ can be realized by negative gate v_i with input connections $c_{j_1,i},\ldots,c_{j_{s_i},i}$.

It only remains to show that N' realizes the correct network output functions. By Step 2 of the algorithm, all specified 0's and 1's on each network output function, $f_1,\ldots,f_m$, become specified 0's and 1's respectively, of $G_c(c_{n+1,n+R+1}),\ldots,G_c(c_{n+m,n+R+m})$. Later, by Step 5, the selected $G_c(v_{n+1})$, $\ldots,G_c(v_{n+m})$ $(f'(v_{n+1}),\ldots,f'(v_{n+m}))$ can only consist, respectively, of subsets of the sets of permissible functions $G_c(c_{n+1,n+R+1}),\ldots,G_c(c_{n+m,n+R+m})$. Hence, $f'(v_{n+1}),\ldots,f'(v_{n+m})$ will contain all specified 0's and 1's of $f_1,\ldots,$ f_m, respectively (and since $v_{n+1},\ldots,v_{n+m}$ are connected to output terminals $v_{n+R+1},\ldots,v_{n+R+m}$, respectively, these output functions are available at the output terminals).

Q.E.D.

After some consideration of Algorithm PP, certain possible modifications of the pruning procedure suggest themselves. These modifications could result inthe enlargement of the CSPF's determined by the algorithm. Larger CSPF's generally allow a greater possibility of pruning connections (fewer 1-0 covers are required). Also, they are generally more desirable when the results of Algorithm PP are to be used in subsequent network transformations.

One such modification consists of altering the assignment of 1-0 covers in Step 7. Presently, a 1-0 cover for a 0-1 pair is always assigned to the input of highest preference according to ordering σ_3 among those inputs for which 1-0 covers exist. However, it is easily possible that a less preferred input connection has previously been assigned the necessary 0 and necessary 1 which constitute a 1-0 cover for that 0-1 pair. This further suggests assigning all essential 1-0 covers before all others. Detection of such existing assignments of 1-0 covers for less preferred input connections can allow the unnecessary assignment of a 1-0 cover to the most preferred connection to be skipped.

Example 6.1.4 Such a situation is shown in Fig. 6.1.10 ($\{v_{j_1}, v_{j_2}, v_{j_3}\} = IP(v_i)$ and $\sigma_3(v_{j_1}) < \sigma_3(v_{j_2}) < \sigma_3(v_{j_3})$ are assumed). As currently stated, Algorithm PP would assign $G_c^{(3)}(c_{j_2,i}) = 0$ and $G_c^{(5)}(c_{j_2,i}) = 1$ to cover the 0-1 pair $G_c^{(5)}(v_i) = 0$, $G_c^{(3)}(v_i) = 1$. With the suggested modification, $G_c^{(3)}(c_{j_3,i}) = 0$ and $G_c^{(5)}(c_{j_3,i}) = 1$ have first been assigned since they are involved in 1-0 covers for 0-1 pairs $G_c^{(2)}(v_i)$, $G_c^{(3)}(v_i)$ and $G_c^{(5)}(v_i)$, $G_c^{(4)}(v_i)$, respectively. Hence, the assignments of $G_c^{(3)}(c_{j_2,i})$ and $G_c^{(5)}(c_{j_2,i})$ to cover the 0-1 pair $G_c^{(5)}(v_i)$, $G_c^{(3)}(v_i)$ can be skipped due to the existence of a previously assigned 1-0 cover. The consequent increase in the number of *'s in the vector for $G_c(c_{j_2,i})$ could lead to more *'s in the vector for $G_c(v_{j_2})$. In turn, this would mean fewer required 1-0 covers for $G_c(v_{j_2})$, and the chances of pruning connections to v_{j_2} would be enhanced.

					CSPF's selected by original version of Algorithm PP			CSPF's selected by Algorithm PP with suggested modification		
	$f(v_{j_1})$	$f(v_{j_2})$	$f(v_{j_3})$	$G_c(v_i)$	$G_c(c_{j_1,i})$	$G_c(c_{j_2,i})$	$G_c(c_{j_3,i})$	$G_c(c_{j_1,i})$	$G_c(c_{j_2,i})$	$G_c(c_{j_3,i})$
1	1	1	0	0	1	1	*	1	1	*
2	0	0	1	0	*	*	1	*	*	1
3	0	0	0	1	0	0	0	0	*	0
4	0	1	0	1	0	*	0	0	*	0
5	0	1	1	0	*	1	1	*	*	1
6	1	0	0	1	*	0	0	*	0	0

$\sigma_3(v_{j_1}) < \sigma_3(v_{j_2}) < \sigma_3(v_{j_3})$

Fig. 6.1.10 Illustration of effect of modified Algorithm 6.1.1 (PP) in Example 6.1.4. $\{v_{j_1}, v_{j_2}, v_{j_3}\} = IP(v_j)$.

Along somewhat similar lines, a second modification can be proposed. Suppose gate v_i has immediate predecessors $v_{j_1},\ldots,v_{j_{s_i}}$ in Step 7 of the algorithm. Since these immediate predecessors must follow v_i in ordering σ_1, at this stage $G_c(v_{j_1}),\ldots,G_c(v_{j_{s_i}})$ have not yet been evaluated (in Step 5). It is possible that some of these immediate predecessors are gates which have other output connections whose CSPF's have already been assigned during Step 7 for previous v_i. For example, $G_c(c_{j_1,k})$, $k \neq i$, may already be known. Since it can be seen that 1-0 covers assigned to a gate's output connection are eventually assigned to the gate itself (through Step 5), all existing assignments of necessary 1's and necessary 0's to output connections of $v_{j_1},\ldots,v_{j_{s_i}}$ can actually be considered to be assignments of necessary 1's and 0's to respective $G_c(v_{j_1}),\ldots,G_c(v_{j_{s_i}})$. Combining this information (necessary 0's and 1's assigned for other output connections of v_i's inputs) with the strategy suggested in the first modification of making use of existing 1-0 covers, still further reductions in the number of assigned 1's and 0's are possible. This, in turn, means generally larger sets of CSPF's and a greater possibility of pruning connections.

Example 6.1.5 An example of the benefit of the second modification of Algorithm PP is illustrated by the subnetwork in Fig. 6.1.11. It is assumed that $\sigma_3(v_{j_3}) < \sigma_3(v_{j_2}) < \sigma_3(v_{j_1})$ and that 1-0 covers are being assigned for gate v_i in Step 7. As Algorithm PP is stated, or even if modified in the manner first suggested, connection $c_{j_3,i}$ would be assigned the 1-0 cover for the 0-1 pair $G_c^{(e)}(v_i)$, $G_c^{(d)}(v_i)$ due to the high preference of v_{j_3}. If the second modification of the algorithm is implemented, however, the existence of the previously assigned $G_c^{(d)}(c_{j_1,k}) = 0$ and $G_c^{(e)}(c_{j_1,k}) = 1$ is recognized

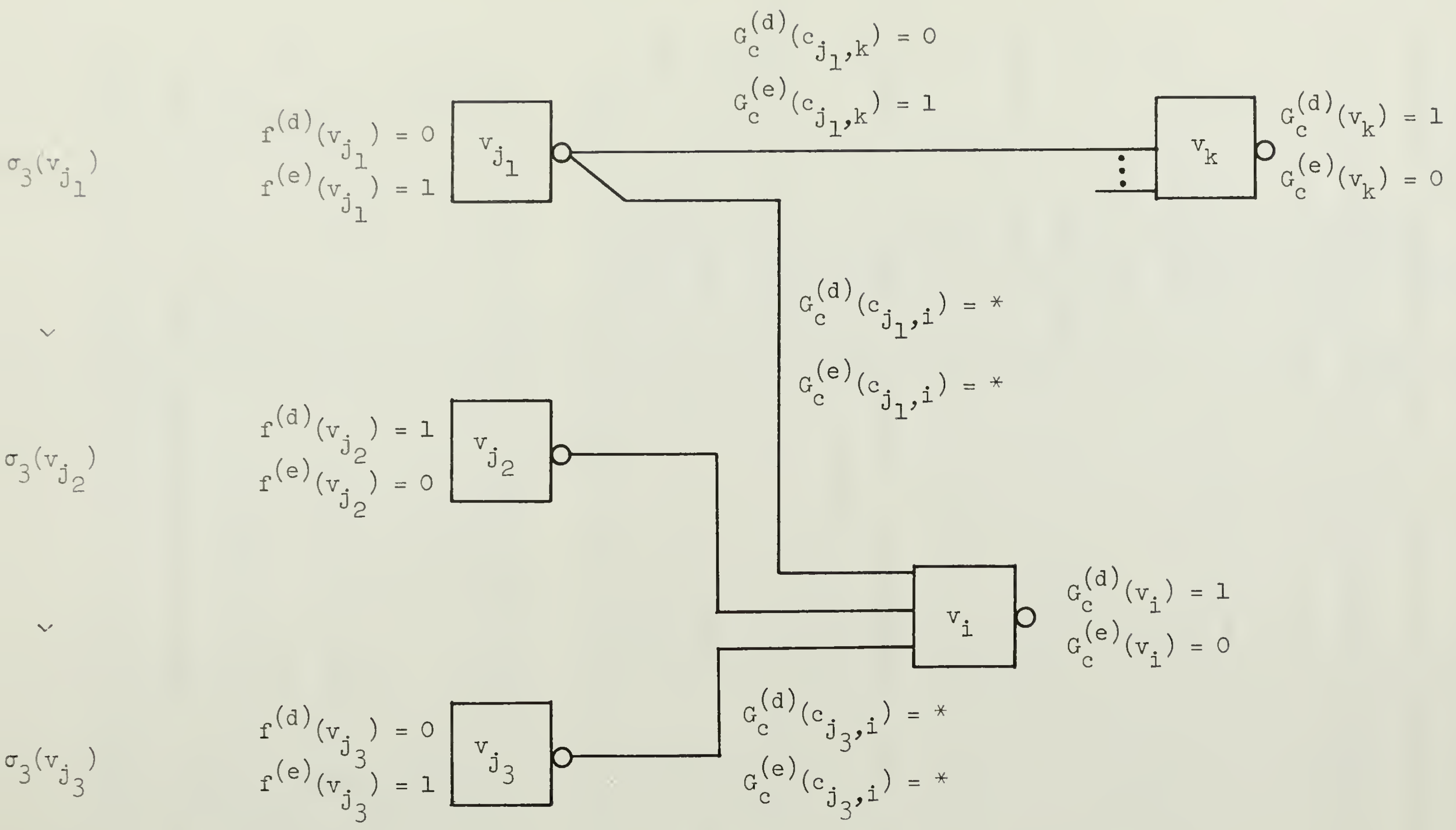


Fig. 6.1.11 Covering situation in subnetwork discussed in Example 6.1.5.

as implying $G_c^{(d)}(v_{j_1}) = 0$, $G_c^{(e)}(v_{j_1}) = 1$. Therefore, $G_c^{(e)}(c_{j_1,i})$, $G_c^{(d)}(c_{j_1,i})$ can be the assigned 1-0 cover for 0-1 pair $G_c^{(e)}(v_i)$, $G_c^{(d)}(v_i)$ (actually, the act of assignment can even be skipped since $G_c^{(d)}(v_{j_1}) = 0$, $G_c^{(e)}(v_{j_1}) = 1$ is already insured) without creating new necessary 0's and 1's in the output function of v_{j_3}.

A third possible modification also improves cover assignments. Basically, this modification exploits the fact that certain necessary 1's and 0's can be predetermined in the CSPF vectors throughout a network which will be present regardless of the preference used in assignmnet of 1-0 covers. The output gates of a network must realize at least the specified 1's and 0's of the corresponding output functions. For certain of the 0-1 pairs among these specified 1's and 0's, essential 1-0 covers will usually exist. Such essential 1-0 covers must eventually be assigned (regardless of the method used)to the respective gates (assignment can be skipped if the essential 1-0 cover belongs to an external variable) as necessary 0's and 1's by Algorithm PP. In turn,these necessary 0's and 1's themselves form 0-1 pairs, of which some will have essential 1-0 covers, etc. As a special step prior to actual pruning (e.g., immediately following Step 2 of the algorithm), such necessary 0's and 1's can first be identified and assigned to vectors representing CSPF's of gates throughout the network. When the pruning procedure is then executed, these pre-assigned necessary 0's and 1's can be used for 1-0 covers whenever possible. If this pre-assignment is not performed, certain 0-1 pairs which can be covered by such (ultimately) necessary 0's and 1's will instead be covered by 1-0 covers assigned to more preferable connections, and, consequently, components of CSPF vectors which might otherwise remain *'s will be assigned to 0's and 1's.

Example 6.1.6 Fig. 6.1.12 illustrates a situation in which the third suggested modification of Algorithm PP would be beneficial. Assume that $\sigma_3(v_\ell) < \sigma_3(v_k)$, $\sigma_1(v_{n+3}) < \sigma_1(v_j)$, and v_{n+1}, v_{n+2}, v_{n+3} are network output gates realizing output functions f_1, f_2, and f_3, respectively. The situation in Fig. 6.1.12 occurs just after Step 2 of the algorithm (for simplicity, assignments made to a gate's output connections will be considered as being made to the gate itself). Suppose that $f^{(b)}(v_j)$, $f^{(a)}(v_j)$ is an essential 1-0 cover of the 0-1 pair $G_c^{(b)}(v_{n+1})$, $G_c^{(a)}(v_{n+1})$ and $f^{(e)}(v_j)$, $f^{(d)}(v_j)$ is an essential 1-0 cover of $G_c^{(e)}(v_{n+2})$, $G_c^{(d)}(v_{n+2})$. Then it is known that assignments $G_c^{(a)}(v_j) = G_c^{(d)}(v_j) = 0$ and $G_c^{(b)}(v_j) = G_c^{(e)}(v_j) = 1$ must eventually be made. Further suppose that $f^{(a)}(v_k)$, $f^{(e)}(v_k)$ is an essential 1-0 cover of $G_c^{(a)}(v_j)$, $G_c^{(e)}(v_j)$, then it is also known that $G_c^{(a)}(v_k) = 1$ and $G_c^{(e)}(v_k) = 0$ will be assigned. Now suppose that Step 7 of Algorithm PP is reached for $v_i = v_{n+3}$, and a 1-0 cover is sought for the 0-1 pair $G_c^{(a)}(v_{n+3})$, $G_c^{(e)}(v_{n+3})$. Since $\sigma_1(v_{n+3}) < \sigma_1(v_j)$, $G_c(v_j)$, and hence $G_c(c_{kj})$, would not yet have been determined by the unmodified algorithm. Thus, even with the first two modifications suggested, the algorithm would select $G_c^{(a)}(v_\ell)$, $G_c^{(e)}(v_\ell)$ (actually $G_c^{(a)}(c_{\ell i})$, $G_c^{(e)}(c_{\ell i})$) as the 1-0 cover to assign. With the addition of the third modification, however, $G_c^{(a)}(v_k) = 1$, $G_c^{(e)}(v_k) = 0$ would be known after the pre-assignment step and could be utilized at this point as the 1-0 cover of $G_c^{(a)}(v_{n+3})$, $G_c^{(e)}(v_{n+3})$.

It is easy to make the pre-assignment step into a systematic procedure. The necessary procedure, to follow Step 2 of Algorithm PP, would be very similar to Steps 3, 4, 5, 7, and 8 of the algorithm – Step 6 being omitted and covers being assigned in Step 7 only when essential 1-0 covers exist. After this pre-

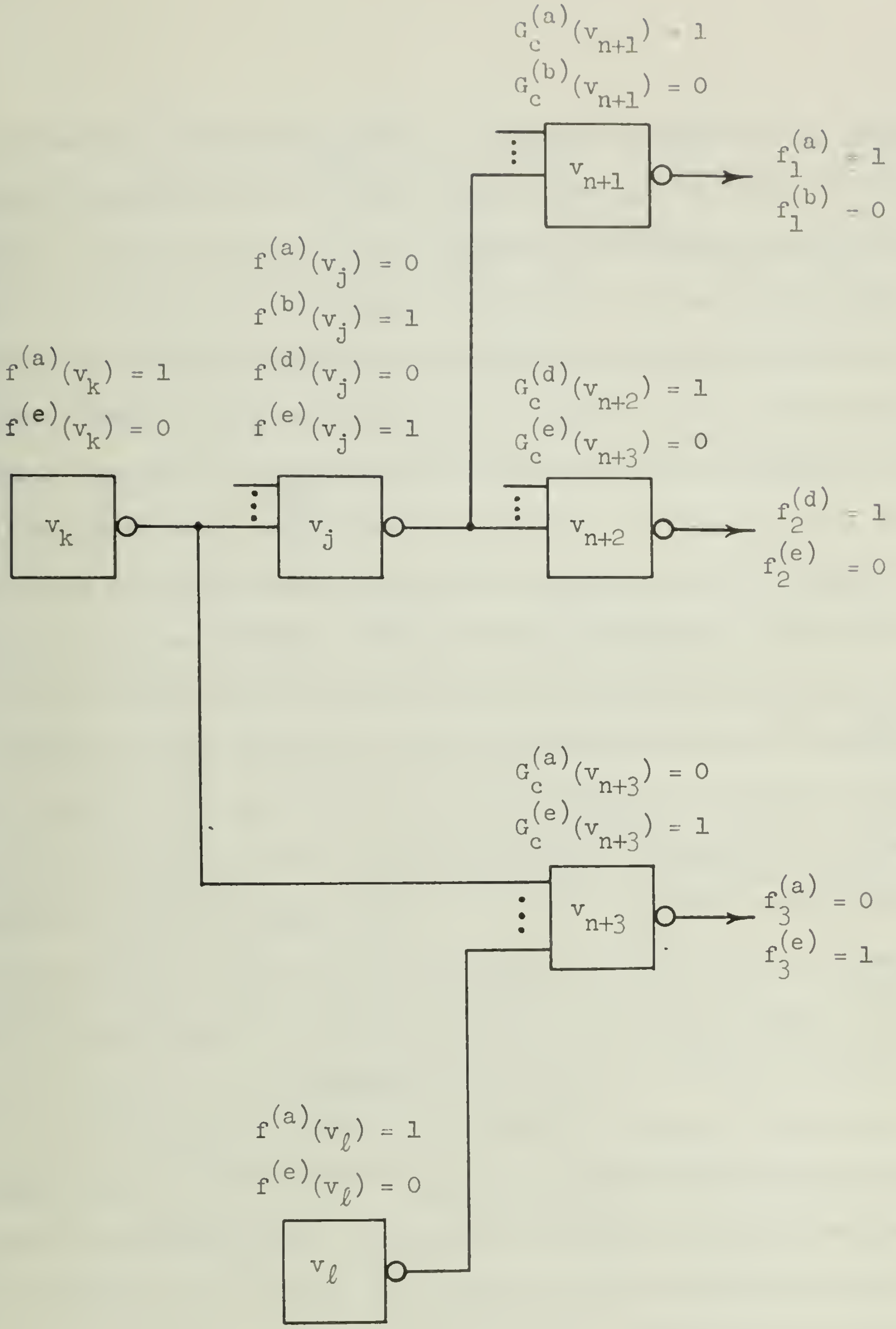


Fig. 6.1.12 Covering situation in subnetwork discussed in Example 6.1.6. $\sigma_3(v_\ell) < \sigma_3(v_k)$ and $\sigma_1(v_{n+3}) < \sigma_1(v_j)$.

assignment of necessary 0's and 1's, the rest of Algorithm PP (with the addition of the second suggested modification to take advantage of the pre-assigned values) would follow normally.

After pruning has been completed by Algorithm PP, or one of its previously described modifications, for a negative gate network, negative completions (i.e., specifications of values for all vectors in V_n) are obtained for the resultant incompletely specified functions, $f'(v_{n+1}),\ldots,f'(v_{n+R})$, realized by gates in N'. Negative completions may be obtained either independently or in combination with the implementations of the negative gates as MOS cells. In either case, pruning may be repeated after a negative completion has been obtained, and it is possible that additional connections may be removed. This occurs since the negative completions may be different from the original functions realized by the gates and may produce new possibilities for cover assignments. Obtaining the configurations of MOS cells implementing the negative gates can involve considerably more calculation than obtaining only simple negative completions, and negative completions are usually sufficient if it is intended to repeat the pruning process. MOS cell configurations can be determined following the last repetition of Algorithm PP.

Since negative completions have been discussed extensively in the literature as related to other synthesis methods [IM 71],[Iba 71],[Lai 76] and since the exact method used is generally incidental to the presentation of transduction procedures, only a simple method will be given here (this method is similar to the one offered in the proof of Theorem 2.4):

Algorithm 6.1.2 Algorithm to obtain negative completions of incompletely specified functions, $f'(v_{n+1}),\ldots,f'(v_{n+R})$, realized by negative gates in a network N' resulting from Algorithm PP (NC).

Functions $x_1,\ldots,x_n$, $f'(v_{n+1}),\ldots,f'(v_{n+R})$ are assumed to be expressed in truth table form.

Step 1 Select a specific ordering σ_1 of gates. Let $f''(v_k) \equiv x_k$, $k = 1, \ldots, n$.

Step 2 Initially, let I be the set of all gates in N'.

Step 3 Select $v_i \in I$ such that $\sigma_1(v_i) > \sigma_1(v)$ for every $v \in I$, $v \neq v_i$.

Step 4 Let $IP(v_i) = \{v_{j_1},\ldots,v_{j_{s_i}}\}$. Form a negative completion, $f''(v_i)$, of $f'(v_i)$ as follows.

(a) Seek a row, d, of the truth table containing entry * in column $v_i (\equiv g_{i-n})$ of the truth table. If none exists, go to Step 5.

(b) If the input vector composed of the d^{th} entries in columns $v_{j_1},\ldots, v_{j_{s_i}}$, $(f''^{(d)}(v_{j_1}),\ldots,f''^{(d)}(v_{j_{s_i}}))$, is less that or equal to any existing true vector of $f''(v_i)$ (i.e., the input vectors corresponding to existing 1 entries in column v_i), assign the * in column v_i the value 1. Otherwise, assign it the value 0. Return to substep (a).

Step 5 $I \Longleftarrow I - \{v_i\}$. If I is not empty, return to Step 3. Otherwise, terminate the algorithm. The negative completions $f''(v_{n+1}),\ldots,f''(v_{n+R})$ are expressed as columns of the completed truth table.

Example 6.1.7 Continuing with the pruning problem discussed in Example 6.1.3, negative completions are obtained by Algorithm NC for the functions realized by gates g_1, g_2, g_3 which are given in the truth table in Fig. 6.1.8. The negative completions are shown in Fig. 6.1.13. Algorithm 6.1.1 (PP) is repeated for the network shown in Fig. 6.1.8 whose gates realize the functions given in Fig. 6.1.13. During this second application of the algorithm, two additional connections are pruned from the network. The resulting network and set of negative gate functions are shown in Fig. 6.1.14. Negative completions are once again obtained by Algorithm NC, giving the truth table in Fig. 6.1.15. From these functions, the MOS cell implementation of the negative gate network of Fig. 6.1.14 is easily obtained. It is also shown in Fig. 6.1.15. A third application of Algorithm 6.1.1 (PP) is found to result in no further pruning (the network in Fig. 6.1.15 actually contains the minimum numbers of MOS cells, intercell connections, and FET's). For comparison, an MOS network implementing the original negative gate network of Fig. 6.1.1 (discussed in Example 6.1.1) is shown in Fig. 6.1.16.

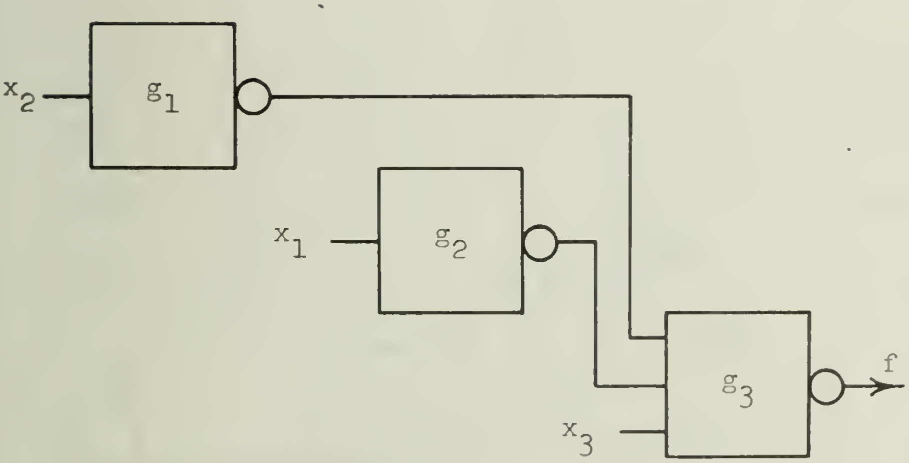
As previously discussed, pruning procedures only remove connections. Next, a transduction procedure will be considered which allows new connections to be added. Let this type of transduction procedure be referred to as 'general' transduction procedures[†] to distinguish them from the pruning type. The general transduction procedure which will be given follows many of the same steps as the pruning procedure Algorithm 6.1.1 (PP). The basic difference is that before the selection of an irredundant input set (Step 6 of Algorithm 6.1.1 (PP)) for a gate v_i, new input connections are added to v_i. These new connections are given lowest priority for removal to maximize the possibility of obtaining (in Step 6) an irredundant input set containing 'fresh' (i.e., previously non-connected) inputs.

Unlike the pruning procedure, the general transduction procedure does not insure a final result having the same or smaller number of connections as the original network. Should such networks occur, they may of course be simply dis-

[†] Comparable transduction procedures for NOR gates are discussed in [Cul 75] and [KC 76].

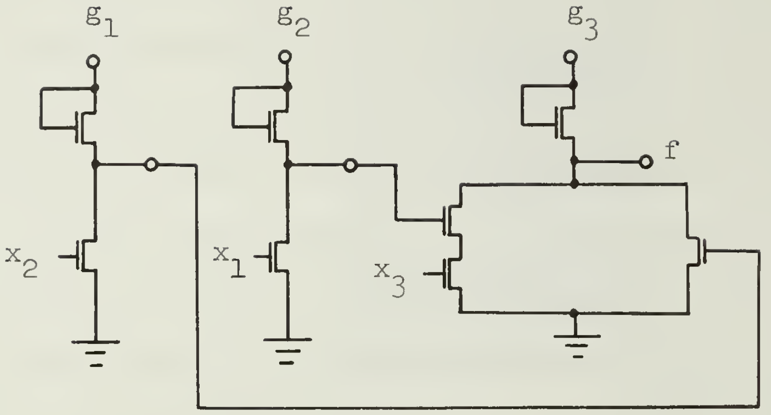
x_1	x_2	x_3	g_1	g_2	g_3
0	0	0	1	1	0
0	0	1	1	1	0
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	0	0	1

Fig. 6.1.13 Negative completions obtained by Algorithm 6.1.2 (NC) for functions in truth table of Fig. 6.1.8. Example 6.1.7.



x_1	x_2	x_3	g_1	g_2	g_3
0	0	0	1	*	0
0	0	1	1	*	0
0	1	0	0	*	1
0	1	1	*	1	0
1	0	0	1	*	0
1	0	0	1	*	0
1	1	0	0	*	1
1	1	1	0	0	1

Fig. 6.1.14 Results of application of Algorithm 6.1.1 (PP) for network of Fig. 6.1.8 whose negative gates realize functions given in Fig. 6.1.13. Example 6.1.7.



x_1	x_2	x_3	g_1	g_2	g_3
0	0	0	1	1	0
0	0	1	1	1	0
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	0	0	1

Fig. 6.1.15 Negative completions for functions in truth table of Fig. 6.1.14 and MOS cell implementation of corresponding negative gate network. Example 6.1.7.

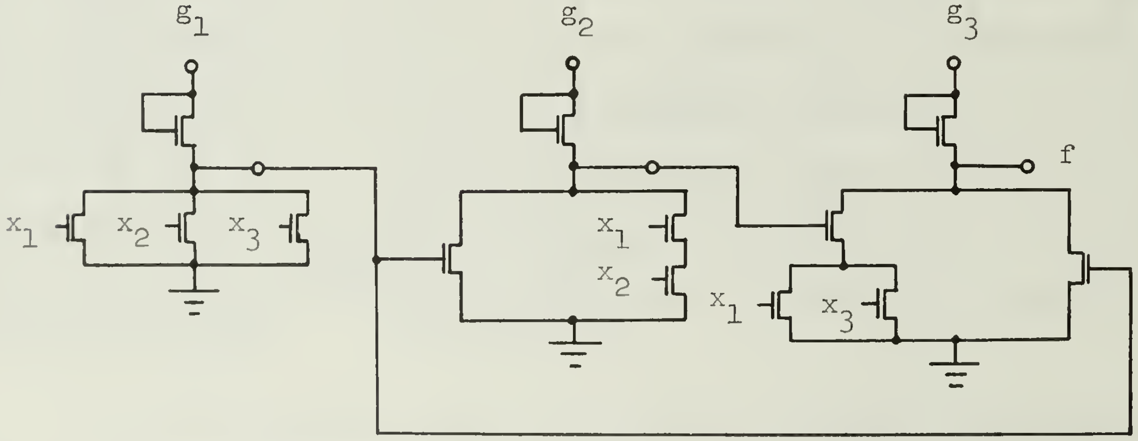


Fig. 6.1.16 MOS cell implementation of original negative gate network of Fig. 6.1.1. For comparison with the MOS network of Fig. 6.1.15 obtained through two applications of Algorithm 6.1.1 (PP) (as discussed in Examples 6.1.1, 6.1.2, 6.1.3, and 6.1.7).

carded, or the general transduction procedure may be repetitively applied for a predetermined number of iterations (obtaining negative completions between iterations) while saving only the best result obtained. Empirical experience has shown that even though the number of connections may increase for an application of the general procedure, a subsequent application (for the resultant network) may decrease the number of connections.

Algorithm 6.1.3 General transduction procedure to add and remove connections from a negative gate network (GP).

A network N of n external variables, m output functions, and R negative gates is transformed into a network N' having the same external variables and output functions.

Step 1 Select a specific ordering σ_1 of gates and orderings σ_2 and σ_3 of gates and external variables (input terminals).

Step 2 Set $G_c(c_{i,i+R}) = f_{i-n}$, $i = (n + 1),\dots,(n + m)$.

Step 3 Initially, let I be the set of all gates in N.

Step 4 Select $v_i \in I$ such that $\sigma_1(v_i) < \sigma_1(v)$ for every $v \in I$, $v \neq v_i$.

Step 5 Compute a CSPF for v_i as follows:

$$G_c(v_i) = \bigcap_{v_j \in IS(v_i)} G_c(c_{ij}).$$

Step 6 Find all connections c_{ji} such that c_{ji} does not currently exist in the network and the addition of c_{ji} would not cause a feed-back loop in the network. Add all such connections to the network. Let $NIP(v_i)$ denote the set of new immediate predecessors of v_i resulting from these connections, and let $OIP(v_i)$ denote the set of old immediate predecessors. ($IP(v_i) = NIP(v_i) \cup OIP(v_i)$.) Form an ordering σ_2' of gates and external variables (input terminals) such that:

$\sigma_2'(v_k) < \sigma_2'(v_\ell)$ for every pair $v_k \in OIP(v_i)$, $v_\ell \in NIP(v_i)$;

and

$\sigma_2'(v_k) < \sigma_2'(v_\ell)$ for every pair v_k, v_ℓ satisfying $\sigma_2(v_k) < \sigma_2(v_\ell)$ and either $v_k, v_\ell \in OIP(v_i)$ or $v_k, v_\ell \in NIP(v_i)$.

Step 7 Select an irredundant input set for v_i with respect to $G_c(v_i)$ as follows:

(a) Seek a $v_j \in IP(v_i)$ such that:

(i) there exist no indices d,e satisfying $G_c^{(d)}(v_i) = f^{(e)}(v_j) = 0$; $G_c^{(e)}(v_i) = f^{(d)}(v_j) = 1$; and either $f^{(d)}(v) = 0$ or $f^{(e)}(v) = 1$ for every $v \in IP(v_i)$, $v \neq v_j$; and

(ii) for every other $v_k \in IP(v_i)$ satisfying condition (i), $\sigma_2'(v_j) < \sigma_2'(v_k)$.

(b) If no such v_j exists, continue to Step 8. Otherwise, $IP(v_i) \Longleftarrow IP(v_i) - \{v_j\}$ and return to substep (a).

Step 8 Calculate CSPF's for remaining input connections to v_i (assign 1-0 covers for 0-1 pairs of $G_c(v_i)$). Compatible sets of permissible functions are determined for remaining connections to v_i by the following specification of values:

$G_c^{(d)}(c_{ji}) = 0$ if and only if $f^{(d)}(v_j) = 0$, $G_c^{(d)}(v_i) = 1$ and there exists an index e such that: $f^{(e)}(v_j) = 1$, $G_c^{(e)}(v_i) = 0$, and no v, $v \neq v_j$ satisfies $v \in IP(v_i)$, $\sigma_3(v) < \sigma_3(v_j)$, $f^{(d)}(v) = 0$, and $f^{(e)}(v) = 1$;

$G_c^{(d)}(c_{ji}) = 1$ if and only if $f^{(d)}(v_j) = 1$, $G_c^{(d)}(v_i) = 0$, and there exists an index e such that: $f^{(e)}(v_j) = 0$, $G_c^{(e)}(v_i) = 1$, and no

v, $v \neq v_j$ satisfies $v \in IP(v_i)$, $\sigma_3(v) < \sigma_3(v_j)$, $f^{(d)}(v) = 1$, and $f^{(e)}(v) = 0$;

$G_c^{(d)}(c_{ji}) = *$ otherwise.

<u>Step 9</u> $f^{(d)}(v_i) \Longleftarrow G_c^{(d)}(v_i)$, for $d = 1,\ldots,2^n$.

<u>Step 10</u> $I \Longleftarrow I - \{v_i\}$. If I is not empty, return to Step 4. Otherwise, terminate the algorithm. The vectors $f(v_i)$, $i = (n + 1),\ldots,(n + R)$ define the incompletely specified functions realized by the respective negative gates in the new network N'.

The proof of the validity of Algorithm GP is much the same as that for Algorithm 6.1.1 (PP). The first and second modifications suggested for the pruning procedure are also suitable for Algorithm GP. Comments made for the repetitive application of Algorithm 6.1.1 (PP) are also applicable to Algorithm GP.

It should be noted that once Step 8 of the algorithm has been executed for a gate v_j, v_j can still be used as a new input to connect to other gates and to provide 1-0 covers, but it can not be assigned additional 1-0 covers involving necessary 1's and 0's which are not already specified (in $G_c^{(d)}(v_j)$, or equivalently, in $f^{(d)}(v_j)$ after Step 9). The reason is that 1-0 covers are provided for all existing 0-1 pairs of $G_c^{(d)}(v_j)$ during execution of Step 8 for v_j. If an further necessary 0's or 1's were introduced into vector $G_c^{(d)}(v_j)$ at a later time, there would be no guarantee that the new 0-1 pairs so created would be covered by v_j's inputs. The algorithm excludes this possibility in Step 9 by immediately changing $f(v_j)$ to $G_c(v_j)$ after assigning all 1-0 covers for v_j in Step 8. This can be seen to prevent Step 8 from later assigning additional necessary 0's and 1's to $G_c(c_{ji})$ (and hence to $G_c(v_j)$ and $f(v_j)$).

Based on consideration of the two transduction procedures presented above, many variations and extensions can be suggested which emphasize different aspects of the transduction process to produce different results (some, for different objectives). A discussion of these will be left to another occasion.

Except for relatively small problems, Algorithms 6.1.1 (PP) and 6.1.3 (GP) require too much computation to be practical for hand calculation. For this reason, the algorithms have been implemented in a computer program to demonstrate and test the feasibility of using the algorithms for larger problems. The computer program implementation and experimental results will be discussed in the following Section 6.2.

6.2 Computer Program Implementation of Transduction Procedures

A computer program called MOSTRA (for MOS network TRAnsduction) has been coded in FORTRAN for IBM 360/75J computer to implement Algorithms 6.1.1 (PP) and 6.1.3 (GP) of the preceding section. The program occupies approximately 250K bytes of memory and, with certain exceptions, can handle problems involving networks with up to 10 external variables, 10 MOS cells, and 10 output functions.

It is intended that the current program will be expanded at a later time to increase its flexibility. As currently structured, the program accepts a set of output functions for which it first synthesizes, according to the method of [NTK 72], an initial negative gate network. Transduction procedures are then applied to this initial network and the networks subsequently derived from it. This structure was selected for its convenience during experimentation with the program, i.e., initial networks did not need to be developed by hand. For many practical applications of the program, it should be modified to accept pre-designed networks (such a modification is not difficult). When program-designed initial networks are desired, it is also possible to incorporate additional network synthesis methods into the program which would afford the user a choice of initial networks.

Also, further transduction procedures can be added, and a choice of sequences in which procedures are applied can be made available for the user. Currently, the user can only choose between repetitive application of the pruning procedure and repetitive application of the general procedure (i.e., no combination of the two is available). The selected procedure is applied repetitively, beginning with the original network, until the network cost (defined as 1000 × (the number of gates) plus 1 × (the number of connections)) no longer decreases, or for a minimum of two applications.

The output of MOSTRA can be determined by the user to be either in the form of a negative gate or MOS cell network. In the former case, the output consists of a listing of connections among gates and external variables and a truth table expressing the functions realized by all gates of the network. For the latter case, in addition to listing intercell connections and cell output functions (in truth table form), configurations of the cell's drivers are printed out in pictorial form.

Section 6.2.1 will discuss the organization of the program. Following this, the preparation of input data for the submission of synthesis problems to the program is given in Section 6.2.2. Finally, the results of the programmed transduction procedures for certain selected problems will be discussed in Section 6.2.3.

6.2.1 Program organization

MOSTRA consists of 12 subroutines: ASGNCV, COMPR, CONFIG, GENER1, IRINPT, MAIN, NEGCOM, OUTPUT, PRUNE1, STEPZ (a system-supplied timing routine), SUBNET, TESTSQ. The general organization of the program is shown in Fig. 6.2.1.1. An arrow from block i to block j denotes the fact that the subroutine represented by block i calls the subroutine represented by block j.

The functions performed by the different subroutines are as follows:

MAIN (627 cards; 574 FORTRAN statements) reads input data for each problem, checks it for errors and inconsistencies, stores the required output functions, synthesizes an initial network realizing the functions, and prints heading information for the problem and the synthesized initial network. Subroutine TESTSQ is then called.

TESTSQ (53 cards; 43 FORTRAN statements) is called by MAIN to apply different sequences of transduction procedures for testing purposes. Logically, the subroutine is an extension of MAIN and the two can be merged after the development of the program is considered complete. The separation of this code from MAIN allows the program to be changed without the necessity of recompiling MAIN.

PRUNE1 (109 cards; 81 FORTRAN statements) controls the execution of the pruning procedure. Arrays are initialized, and cells are ordered for consideration by the pruning procedure. For each cell, an irredundant input set is first chosen by calling IRINPT. Then, covers are assigned among these inputs by calling ASGNCV. Finally, the resulting CSPF's and intercell connections are printed. The implementation of the pruning procedure incorporates the first two suggested modifications of Algorithm 6.1.1 (PP).

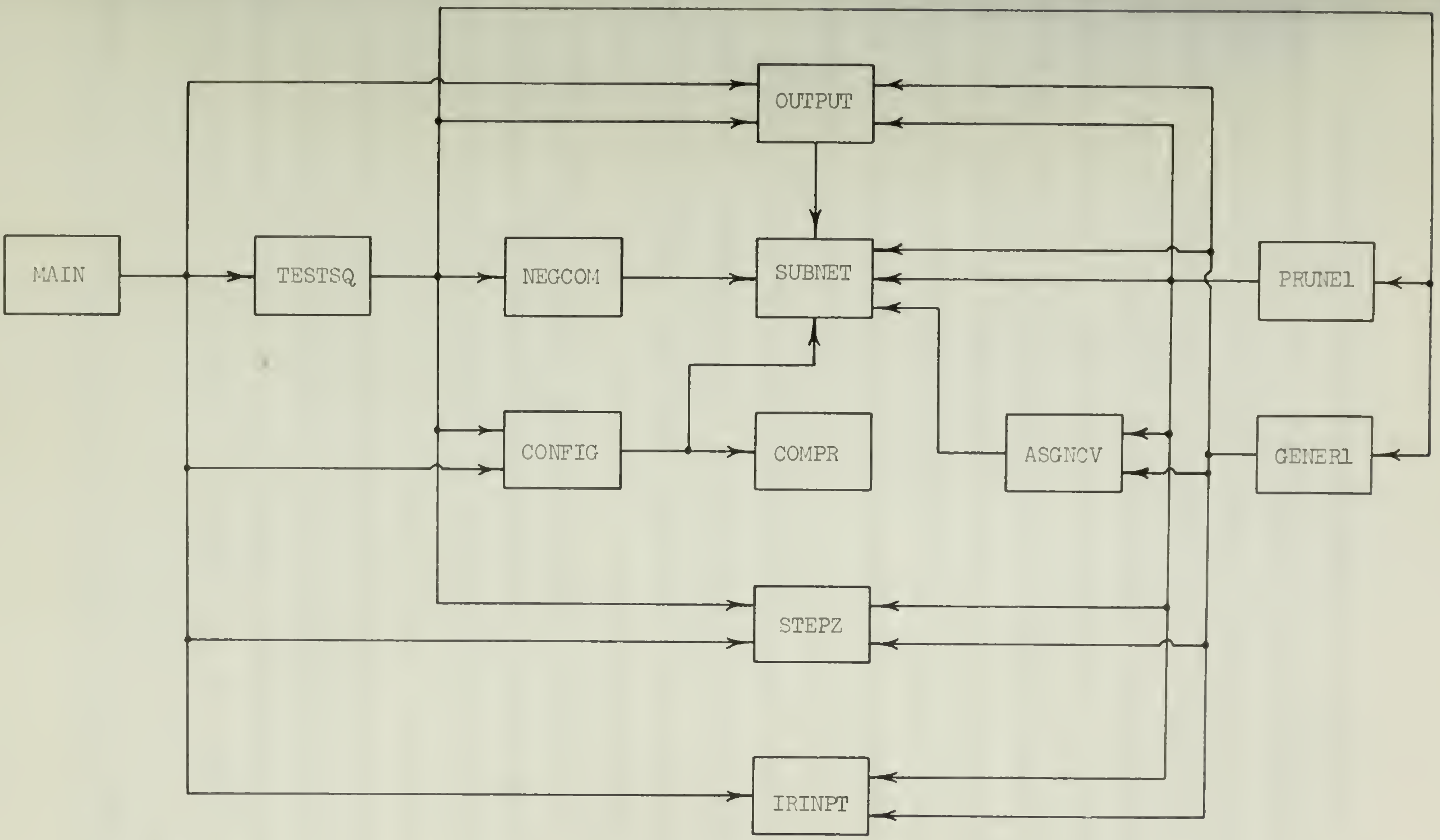


Fig. 6.2.1.1 General organization of program MOSTRA.

GENER1 (124 cards; 108 FORTRAN statements) similarly controls the execution of the general transduction procedure. The ordering in which gates are considered for a reevaluation of their input sets (i.e., ordering σ_1) is slightly different from that used in PRUNE1 in order to promote the exhange of input connections during the transformation of the network. According to this ordering, each cell v_i is selected in turn. For each v_i: new input connections are added to v_i where possible; an irredundant input set is then chosen by calling IRINPT with preference given to retaining newly added connections; and covers are assigned remaining inputs by calling ASGNCV. The resulting CSPF's and intercell connections are printed. Due to the coding adopted for the implementation of the pruning procedure which is shared by the general procedure, it is not convenient to allow a gate v_i, for which Step 8 of Algorithm 6.1.3 (GP) has already been performed, to later be used as the source of new input connections for other gates. Hence, such types of new connections are currently prohibited by the subroutine.

IRINPT (564 cards; 537 FORTRAN statements) determines an irredundant input set for a given gate v_i, and corresponding CSPF $G_c(v_i)$. An essential set of inputs is first determined among existing inputs to v_i. Non-essential inputs are disconnected one-by-one, reevaluating the set of essential inputs between removals (some non-essential input may become essential after the disconnection of each non-essential input). The resultant set of inputs is irredundant in the sense that the removal of any one or more of them would leave at least one 0-1 pair of $G_c(v_i)$ without a 1-0 cover (i.e., v_i would not be a negative gate).

ASGNCV (489 cards; 439 FORTRAN statements) is called by both PRUNE1 and

GENERl for use in both transduction procedures. It assigns 1-0 covers to the inputs of a given gate v_i for 0-1 pairs in vector $G_c(v_i)$ representing the CSPF for v_i. Existing necessary 0's and 1's which form 1-0 covers (thus implementing the first two suggested modifications of Algorithm 6.1.1 (PP) which are also applicable to Algorithm 6.1.3 (GP)) and external variables are the most preferred covers. For 0-1 pairs without such covers, assignments of 1-0 covers are made according to one of two preference orderings selected by the user: (1) a gate in a higher level is preferred to a gate in a lower level or (2) a gate in a lower level is preferred to a gate in a higher level. The user can select the preference ordering desired for each problem by specifying a parameter on the input (data) cards.

NEGCOM (401 cards; 333 FORTRAN statements) given the set of intercell connections of a network and a set of incompletely specified functions realized by the negative gates of the network (each function for a gate v_i is assumed to be negative with respect to the functions realized by v_i's inputs), determines a negative completion of each cell's output function with respect to its inputs. These negative completions are internally expressed in truth table form (2^n rows) and are printed. Speed and the judicious use of memory space are emphasized.

CONFIG (372 cards; 349 FORTRAN statements) is similar to NEGCOM in that it also obtains negative completions of the incompletely specified negative functions to be realized by the negative gates of the network. In addition, however, actual cell configurations corresponding to the negative completions are constructed and printed out. Each cell's configuration is irredundant with respect to the incompletely specified function for which it is synthesized (i.e., no FET's can be shorted or removed from an individual cell without causing it to realize a function which is not in the CSPF of that cell). Most of this subroutine is taken from subroutine IMC of the program DIMN developed by K. Yamamoto [Yam 76] to implement network synthesis methods of [Lai 76]. Subroutine IMC requires the use of subroutine COMPR which has also been included in the current program.

COMPR (9 cards; 12 FORTRAN statements) is called by CONFIG to compare the size of two vectors.

OUTPUT (139 cards; 130 FORTRAN statements) has three entry points: CKT prints out a table of intergate connections and the level of each gate in the network, calculates and prints the 'cost' of the network (1000 × (the number of gates) plus 1 × (the number of intergate connections)),and prints the functions realized by the gates in truth table form; NTCOST calculates, but does not print, the cost of the network; TRUTH prints the functions realized by the cells in truth table form.

SUBNET (99 cards; 102 FORTRAN statements) has three entry points: SUCCES determines the successors of all gates and external variables in the network; PRESUC determines, in addition, the sets of immediate predecessors and immediate successors of each gate and external variable; and LEVELS determines the level of each gate in the network and removes connections among cells in isolated subnetworks. Isolated cells are assigned the level one.

STEPZ is a system-supplied timing routine which, when called, returns the number of centiseconds remaining in the time allotted for the job.

The following definitions are made for the convenience of the presentation of subroutine flowcharts:

Definition 6.2.1.1 The 1 and 0 values constituting a 1-0 cover of a 0-1 pair are referred to as a 1 half-cover and a 0 half-cover, respectively, with respect to the 0-1 pair.

Flowcharts of the more important subroutines in the program, MAIN, TESTSQ, PRUNE1, GENER1, IRINPT, and ASGNCV, are given in Figs. 6.2.1.2, 6.2.1.3, 6.2.1.4, 6.2.1.5, 6.2.1.6, 6.2.1.7, respectively.

6.2.2 Input format

This section will describe the preparation of input data acceptable to the program MOSTRA.

Each problem to be submitted is specified on a set of data cards consisting of four groups:

(1) A heading card;

(2) A problem parameter card;

(3) External variable cards; and

(4) Output function card(s).

If several problems are to be submitted, the corresponding sets of data cards are simply arranged sequentially as shown in Fig. 6.2.2.1. The information to appear on the four different groups of data cards and the respective formats are described in the following:

Heading card The contents of this card are read and printed out along with a problem number assigned by the program (the program sequentially numbers problems in the same run). This card enables the user to provide identifying or descriptive information to appear on the printed output.

Problem parameter card The first 40 columns of this card are divided into ten fields of four columns each. The fields should contain the following numeric or symbolic data, right-justified within the respective fields:

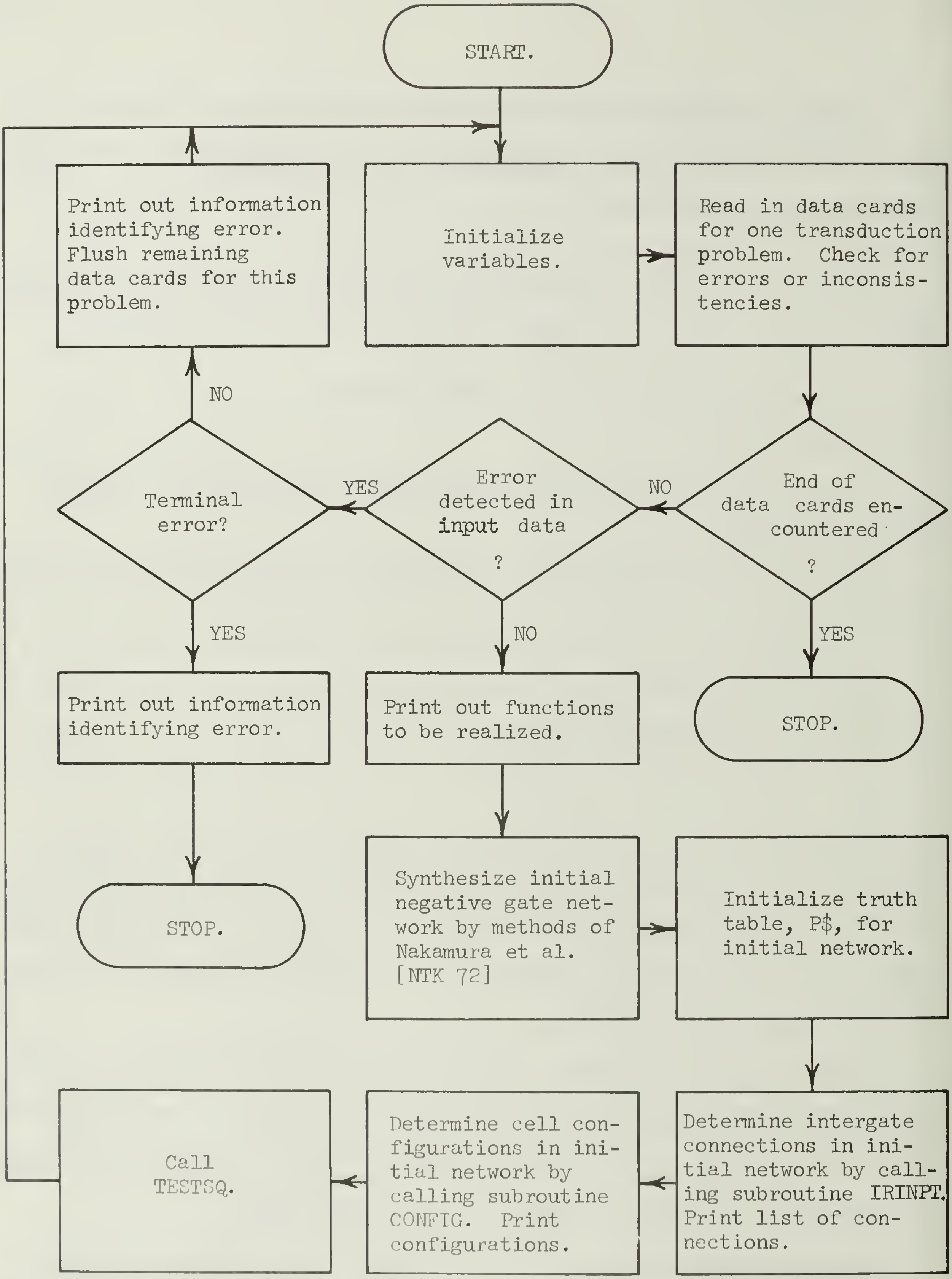

START.
Initialize variables.
Read in data cards for one transduction problem. Check for errors or inconsistencies.
End of data cards encountered ?
YES
STOP.
NO
Error detected in input data ?
YES
Terminal error?
NO
Print out information identifying error. Flush remaining data cards for this problem.
YES
Print out information identifying error.
STOP.
NO
Print out functions to be realized.
Synthesize initial negative gate network by methods of Nakamura et al. [NTK 72]
Initialize truth table, P$, for initial network.
Determine intergate connections in initial network by calling subroutine IRINPT. Print list of connections.
Determine cell configurations in initial network by calling subroutine CONFIG. Print configurations.
Call TESTSQ.

Fig. 6.2.1.2 Flowchart of subroutine MAIN.

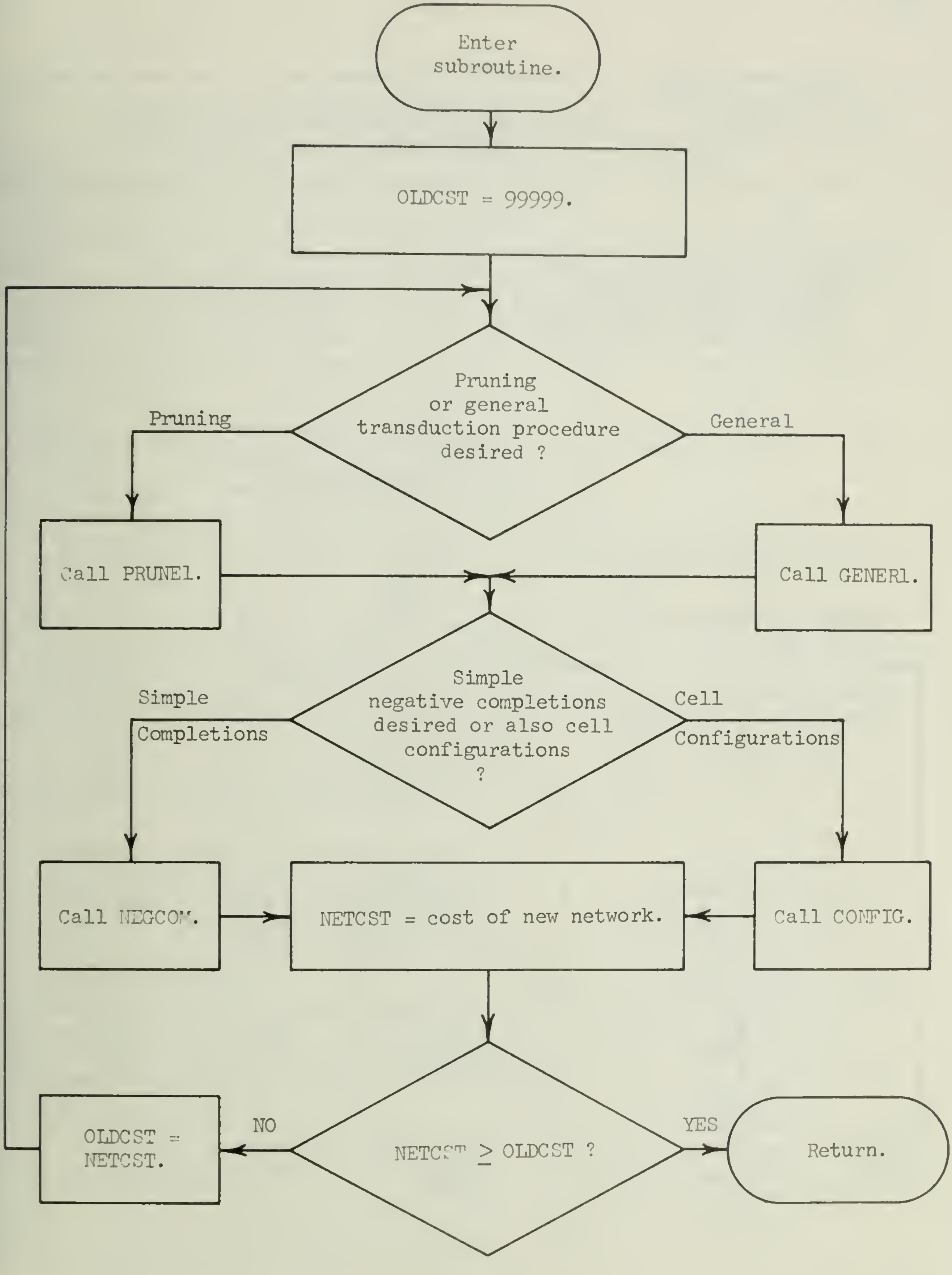


Fig. 6.2.1.3 Flowchart of subroutine TESTSQ.

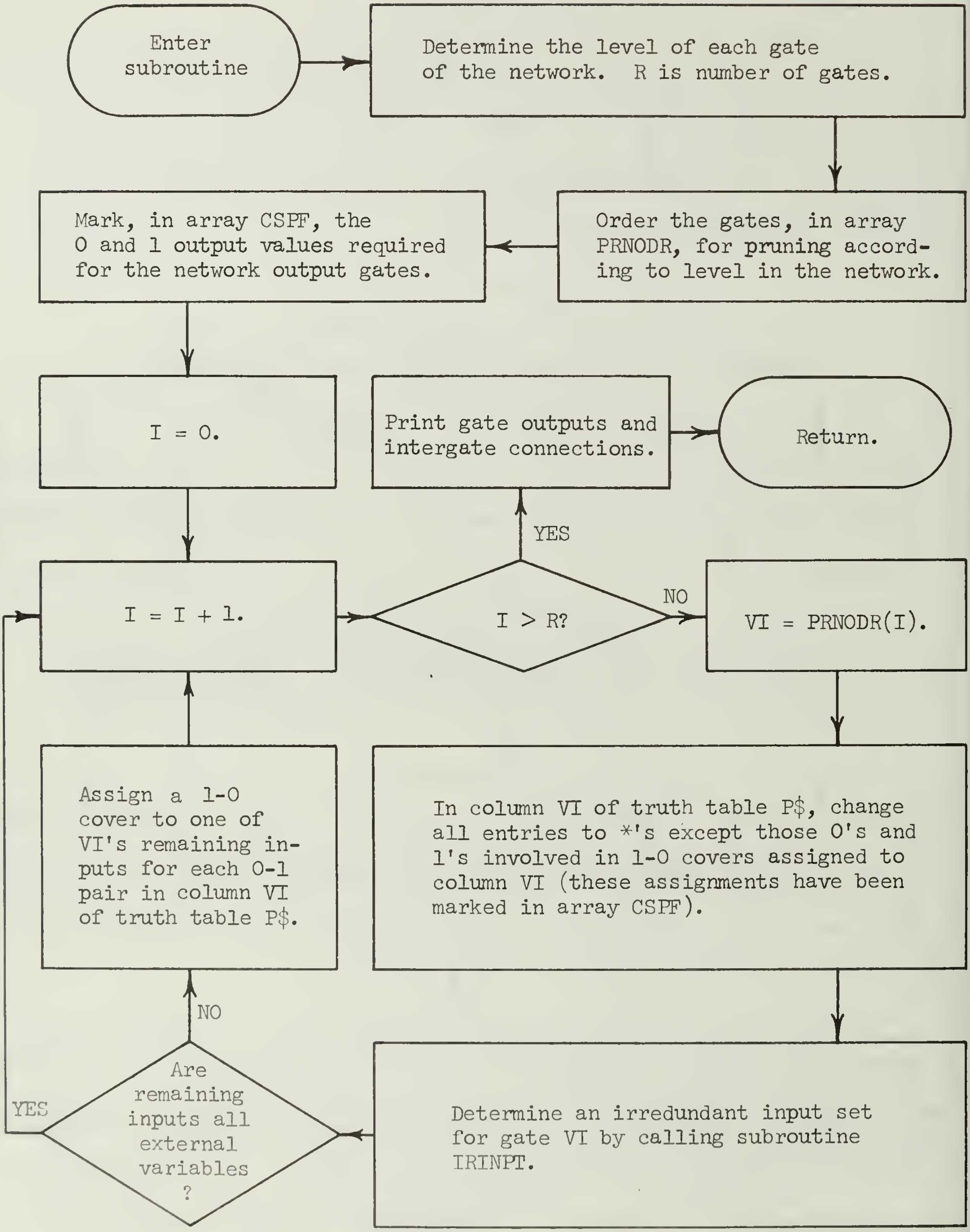

Enter subroutine
Determine the level of each gate of the network. R is number of gates.
Order the gates, in array PRNODR, for pruning according to level in the network.
Mark, in array CSPF, the 0 and 1 output values required for the network output gates.
I = 0.
Print gate outputs and intergate connections.
Return.
YES
I = I + 1.
I > R?
NO
VI = PRNODR(I).
Assign a 1-0 cover to one of VI's remaining inputs for each 0-1 pair in column VI of truth table P$.
In column VI of truth table P$, change all entries to *'s except those 0's and 1's involved in 1-0 covers assigned to column VI (these assignments have been marked in array CSPF).
NO
YES
Are remaining inputs all external variables ?
Determine an irredundant input set for gate VI by calling subroutine IRINPT.

Fig. 6.2.1.4 Flowchart of subroutine PRUNE1.

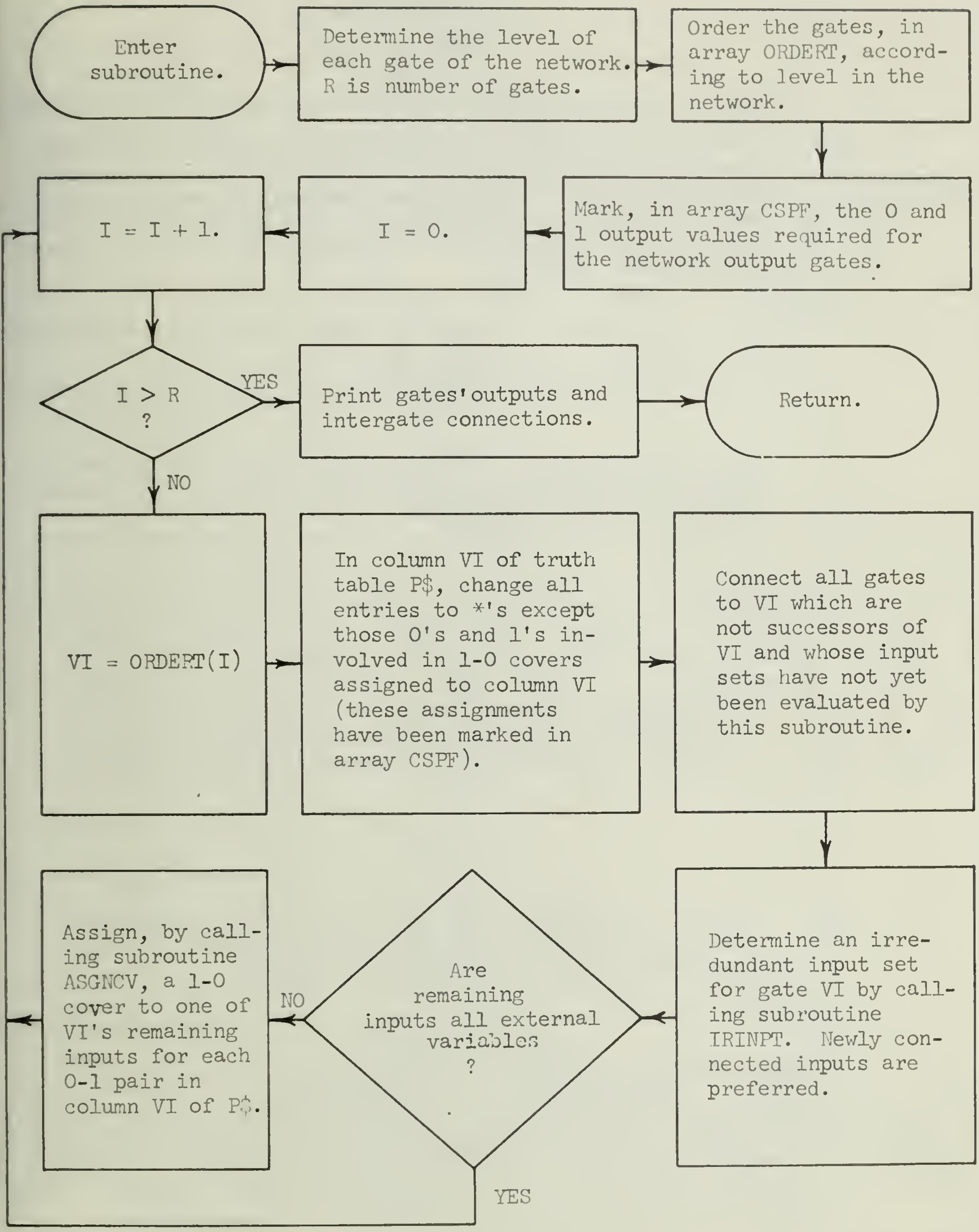

Enter subroutine.
Determine the level of each gate of the network. R is number of gates.
Order the gates, in array ORDERT, according to level in the network.
Mark, in array CSPF, the 0 and 1 output values required for the network output gates.
I = 0.
I = I + 1.
I > R ?
YES
NO
Print gates' outputs and intergate connections.
Return.
VI = ORDERT(I)
In column VI of truth table P$, change all entries to *'s except those 0's and 1's involved in 1-0 covers assigned to column VI (these assignments have been marked in array CSPF).
Connect all gates to VI which are not successors of VI and whose input sets have not yet been evaluated by this subroutine.
Determine an irredundant input set for gate VI by calling subroutine IRINPT. Newly connected inputs are preferred.
Are remaining inputs all external variables ?
NO
YES
Assign, by calling subroutine ASGNCV, a 1-0 cover to one of VI's remaining inputs for each 0-1 pair in column VI of P$.

Fig. 6.2.1.5 Flowchart of subroutine GENER1.

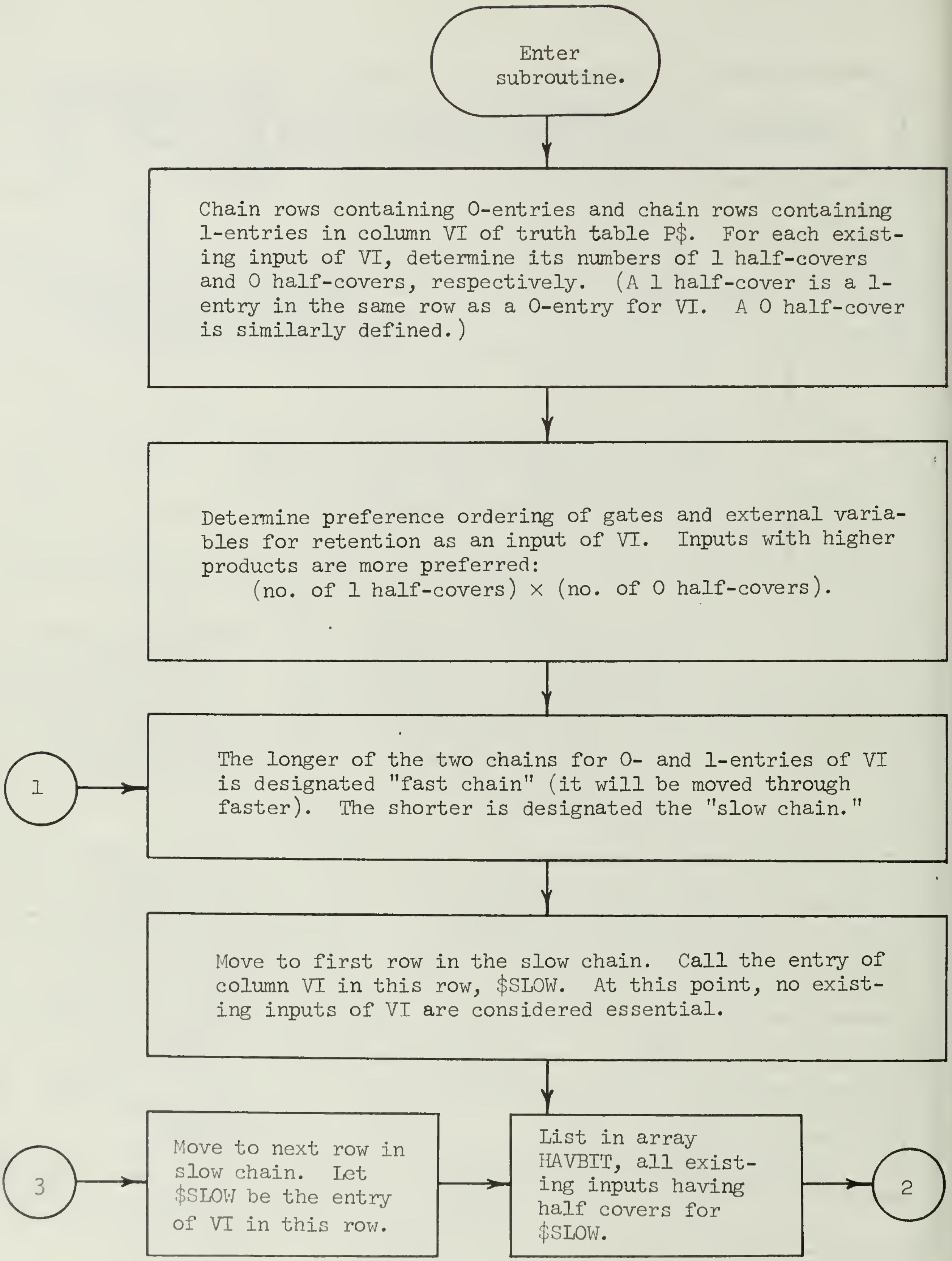

Enter subroutine.
Chain rows containing 0-entries and chain rows containing 1-entries in column VI of truth table P$. For each existing input of VI, determine its numbers of 1 half-covers and 0 half-covers, respectively. (A 1 half-cover is a 1-entry in the same row as a 0-entry for VI. A 0 half-cover is similarly defined.)
Determine preference ordering of gates and external variables for retention as an input of VI. Inputs with higher products are more preferred: (no. of 1 half-covers) × (no. of 0 half-covers).
1
The longer of the two chains for 0- and 1-entries of VI is designated "fast chain" (it will be moved through faster). The shorter is designated the "slow chain."
Move to first row in the slow chain. Call the entry of column VI in this row, $SLOW. At this point, no existing inputs of VI are considered essential.
3
Move to next row in slow chain. Let $SLOW be the entry of VI in this row.
List in array HAVBIT, all existing inputs having half covers for $SLOW.
2

Fig. 6.2.1.6 Flowchart of subroutine IRINPT.

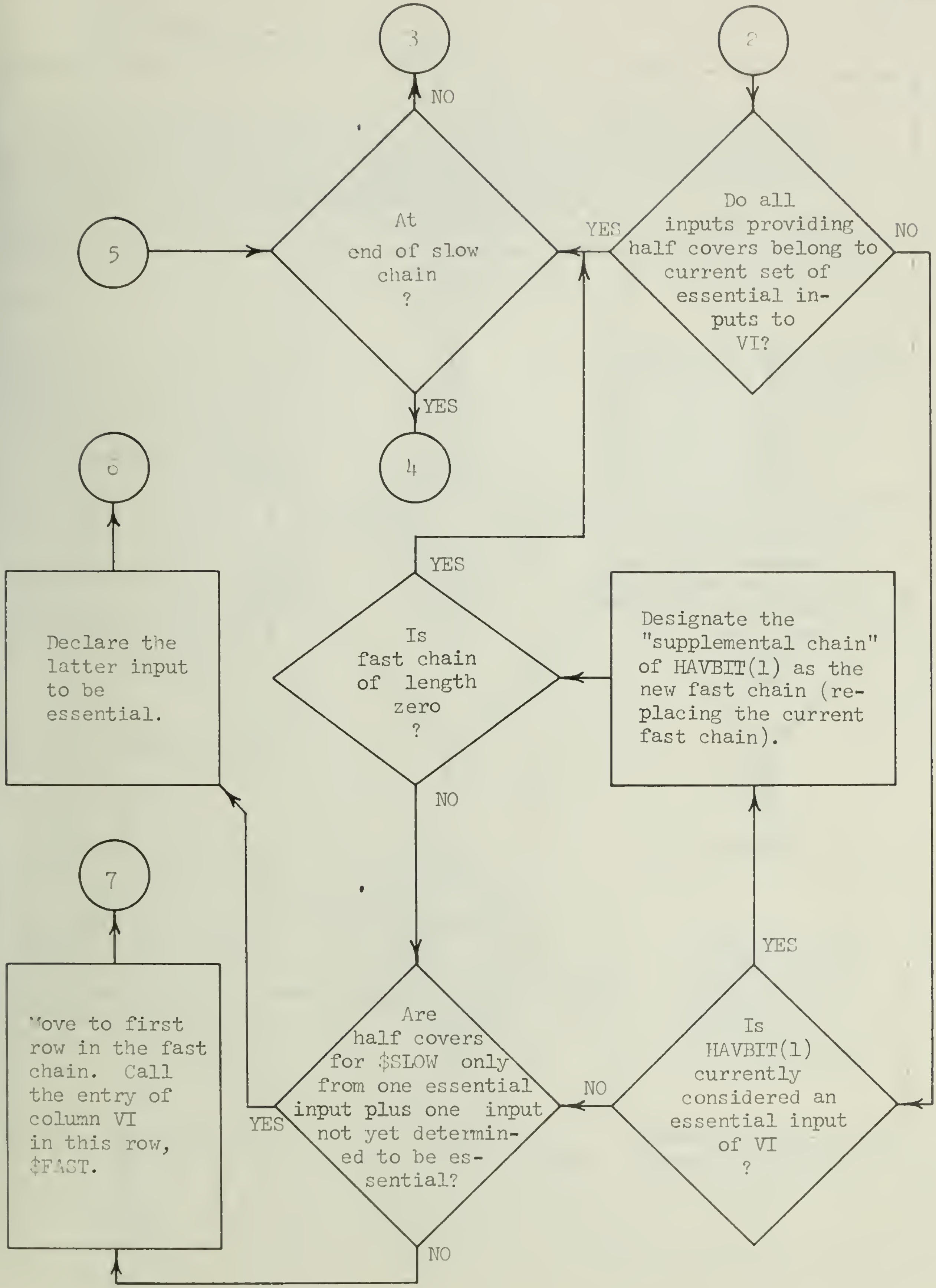

3
2
NO
At end of slow chain ?
YES
Do all inputs providing half covers belong to current set of essential inputs to VI?
NO
5
YES
4
6
YES
Declare the latter input to be essential.
Is fast chain of length zero ?
Designate the "supplemental chain" of HAVBIT(1) as the new fast chain (replacing the current fast chain).
NO
7
YES
Move to first row in the fast chain. Call the entry of column VI in this row, $FAST.
Are half covers for $SLOW only from one essential input plus one input not yet determined to be essential?
YES
NO
Is HAVBIT(1) currently considered an essential input of VI ?
NO

Fig. 6.2.1.6 (Continued)

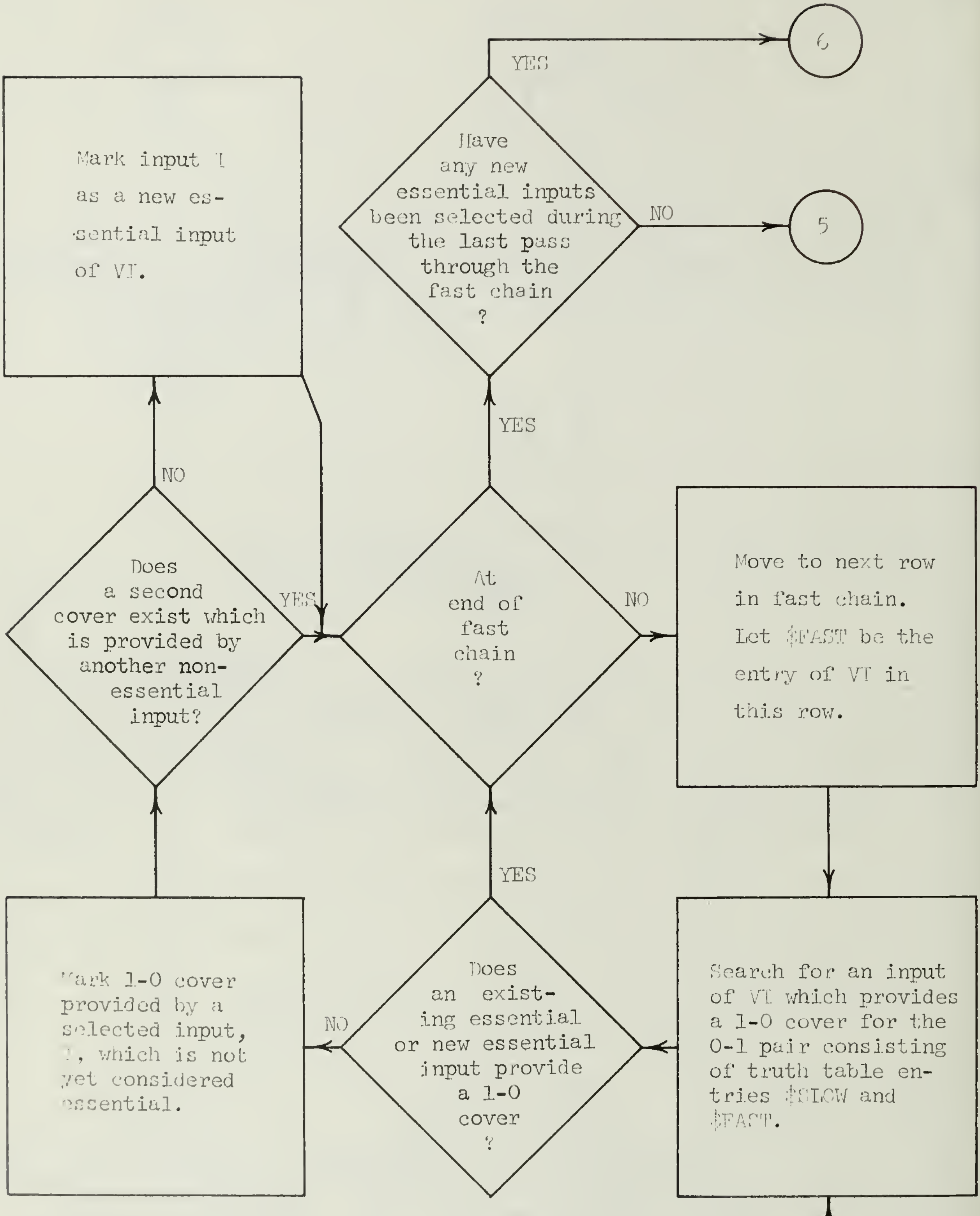

Have any new essential inputs been selected during the last pass through the fast chain ?
YES
NO
6
5
Mark input I as a new essential input of VI.
YES
NO
Does a second cover exist which is provided by another non-essential input?
YES
At end of fast chain ?
NO
Move to next row in fast chain. Let $FAST be the entry of VI in this row.
YES
Mark 1-0 cover provided by a selected input, I, which is not yet considered essential.
NO
Does an existing essential or new essential input provide a 1-0 cover ?
Search for an input of VI which provides a 1-0 cover for the 0-1 pair consisting of truth table entries $SLOW and $FAST.
7

Fig. 6.2.1.6 (Continued)

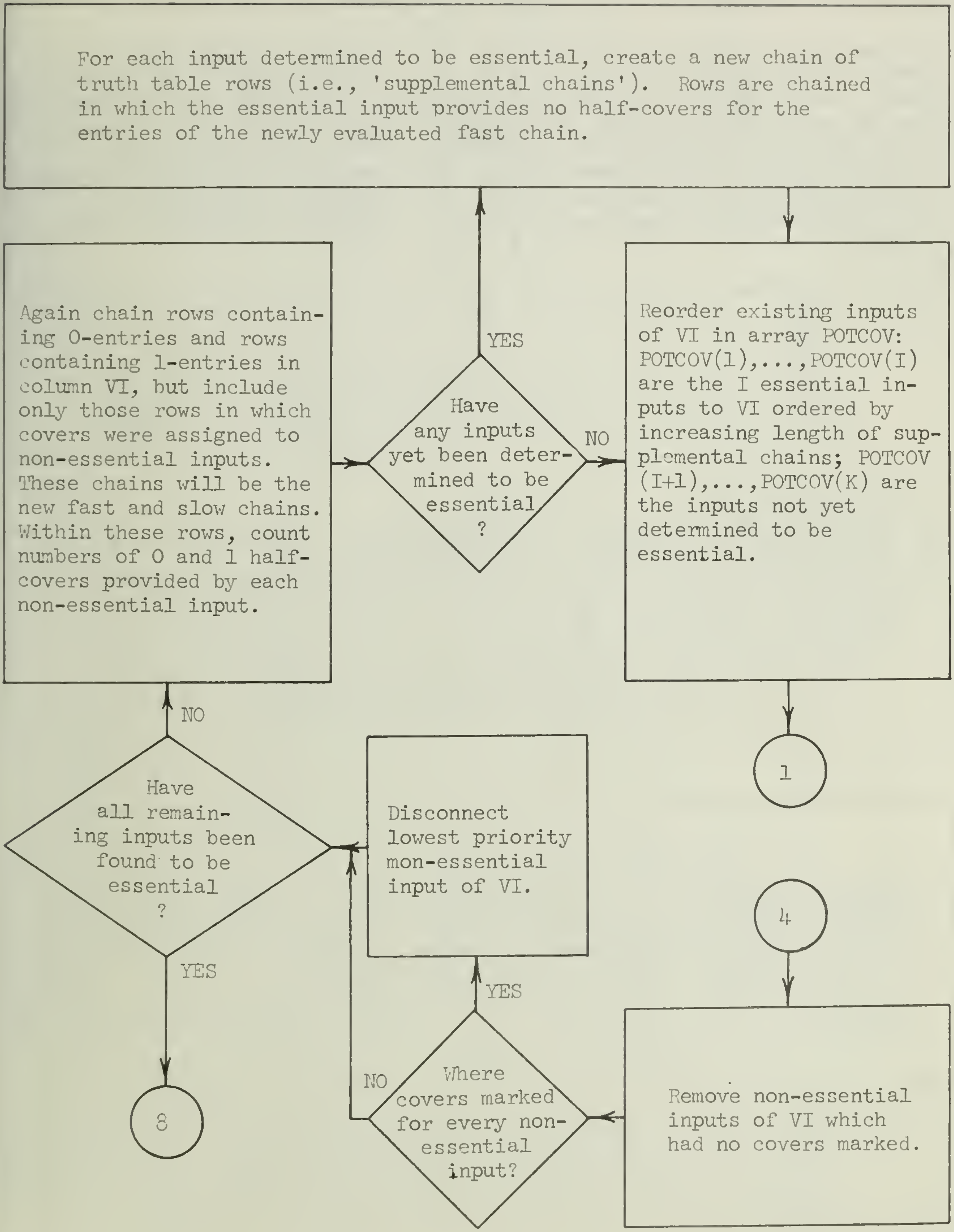

For each input determined to be essential, create a new chain of truth table rows (i.e., 'supplemental chains'). Rows are chained in which the essential input provides no half-covers for the entries of the newly evaluated fast chain.
Again chain rows containing 0-entries and rows containing 1-entries in column VI, but include only those rows in which covers were assigned to non-essential inputs. These chains will be the new fast and slow chains. Within these rows, count numbers of 0 and 1 half-covers provided by each non-essential input.
YES
Have any inputs yet been determined to be essential ?
NO
Reorder existing inputs of VI in array POTCOV: POTCOV(1),...,POTCOV(I) are the I essential inputs to VI ordered by increasing length of supplemental chains; POTCOV (I+1),...,POTCOV(K) are the inputs not yet determined to be essential.
NO
1
Have all remaining inputs been found to be essential ?
Disconnect lowest priority mon-essential input of VI.
4
YES
YES
NO
Where covers marked for every non-essential input?
Remove non-essential inputs of VI which had no covers marked.
8

Fig. 6.2.1.6 (Continued)

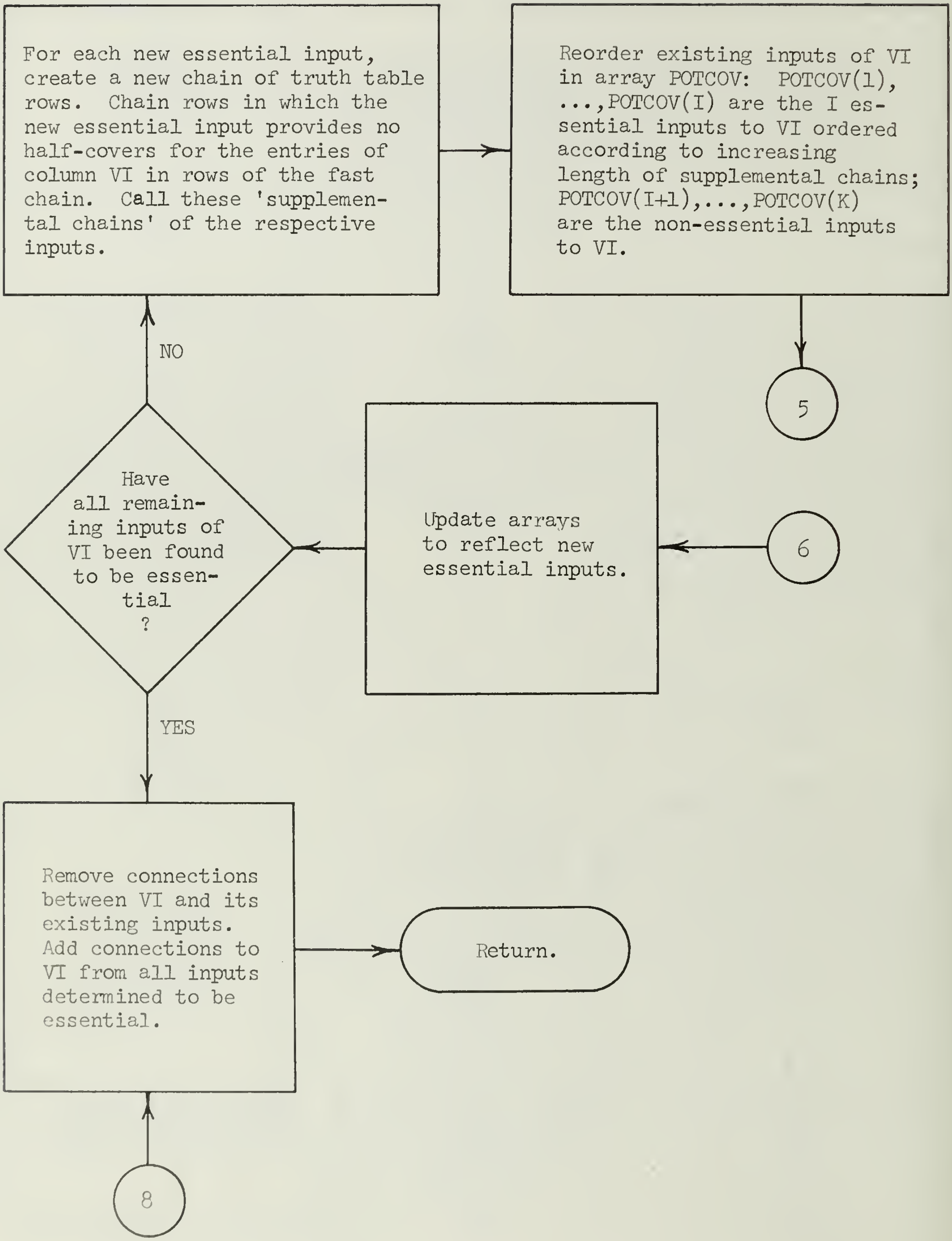
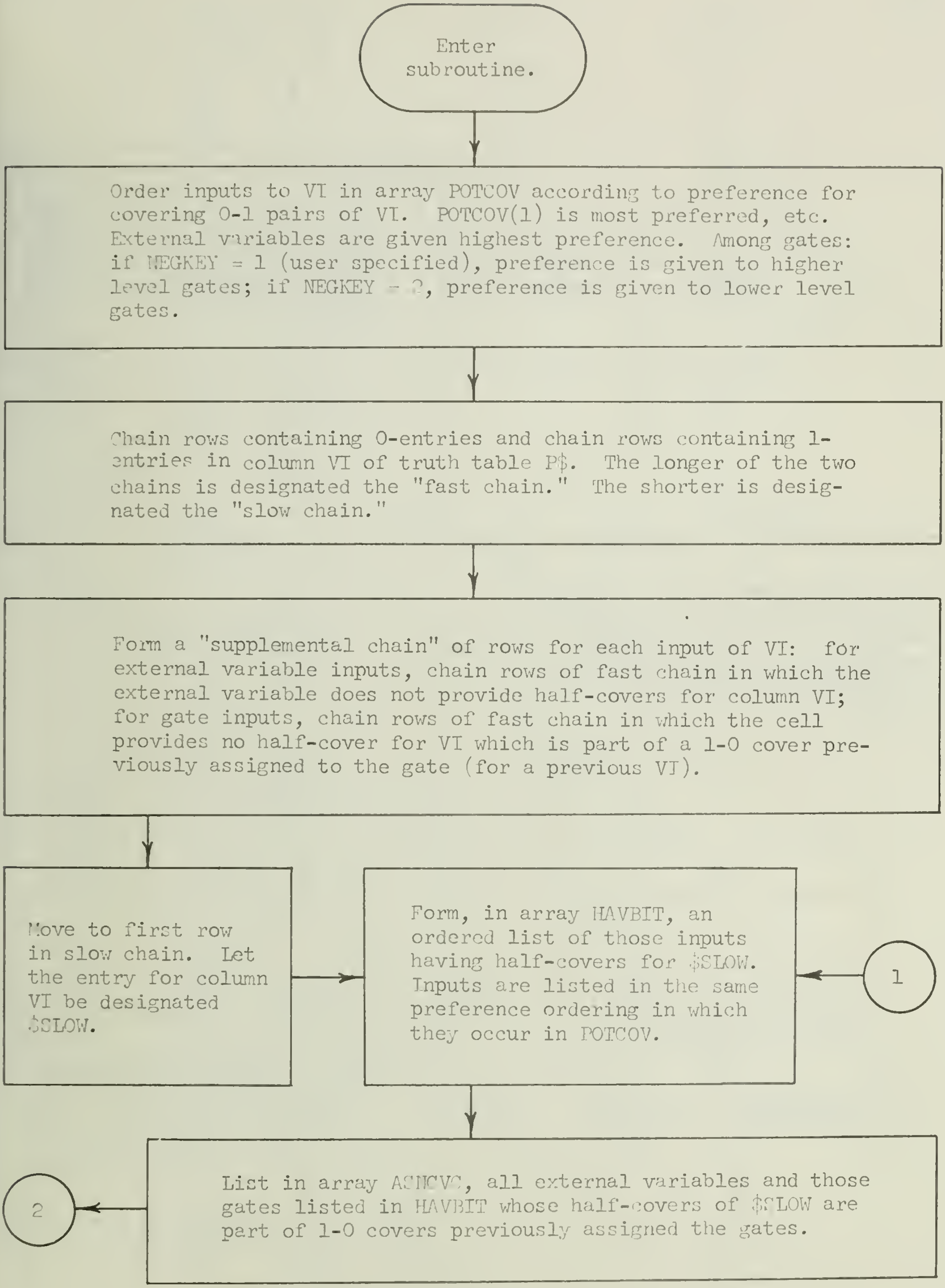

For each new essential input, create a new chain of truth table rows. Chain rows in which the new essential input provides no half-covers for the entries of column VI in rows of the fast chain. Call these 'supplemental chains' of the respective inputs.
Reorder existing inputs of VI in array POTCOV: POTCOV(1), ...,POTCOV(I) are the I essential inputs to VI ordered according to increasing length of supplemental chains; POTCOV(I+1),...,POTCOV(K) are the non-essential inputs to VI.
NO
5
Have all remaining inputs of VI been found to be essential ?
Update arrays to reflect new essential inputs.
6
YES
Remove connections between VI and its existing inputs. Add connections to VI from all inputs determined to be essential.
Return.
8

Fig. 6.2.1.6 (Continued)



Enter subroutine.

Order inputs to VI in array POTCOV according to preference for covering 0-1 pairs of VI. POTCOV(1) is most preferred, etc. External variables are given highest preference. Among gates: if NEGKEY = 1 (user specified), preference is given to higher level gates; if NEGKEY = 2, preference is given to lower level gates.

Chain rows containing 0-entries and chain rows containing 1-entries in column VI of truth table P$. The longer of the two chains is designated the "fast chain." The shorter is designated the "slow chain."

Form a "supplemental chain" of rows for each input of VI: for external variable inputs, chain rows of fast chain in which the external variable does not provide half-covers for column VI; for gate inputs, chain rows of fast chain in which the cell provides no half-cover for VI which is part of a 1-0 cover previously assigned to the gate (for a previous VI).

Move to first row in slow chain. Let the entry for column VI be designated $SLOW.

Form, in array HAVBIT, an ordered list of those inputs having half-covers for $SLOW. Inputs are listed in the same preference ordering in which they occur in POTCOV.

List in array ASNCVC, all external variables and those gates listed in HAVBIT whose half-covers of $SLOW are part of 1-0 covers previously assigned the gates.

Fig. 6.2.1.7 Flowchart of subroutine ASGNCV.

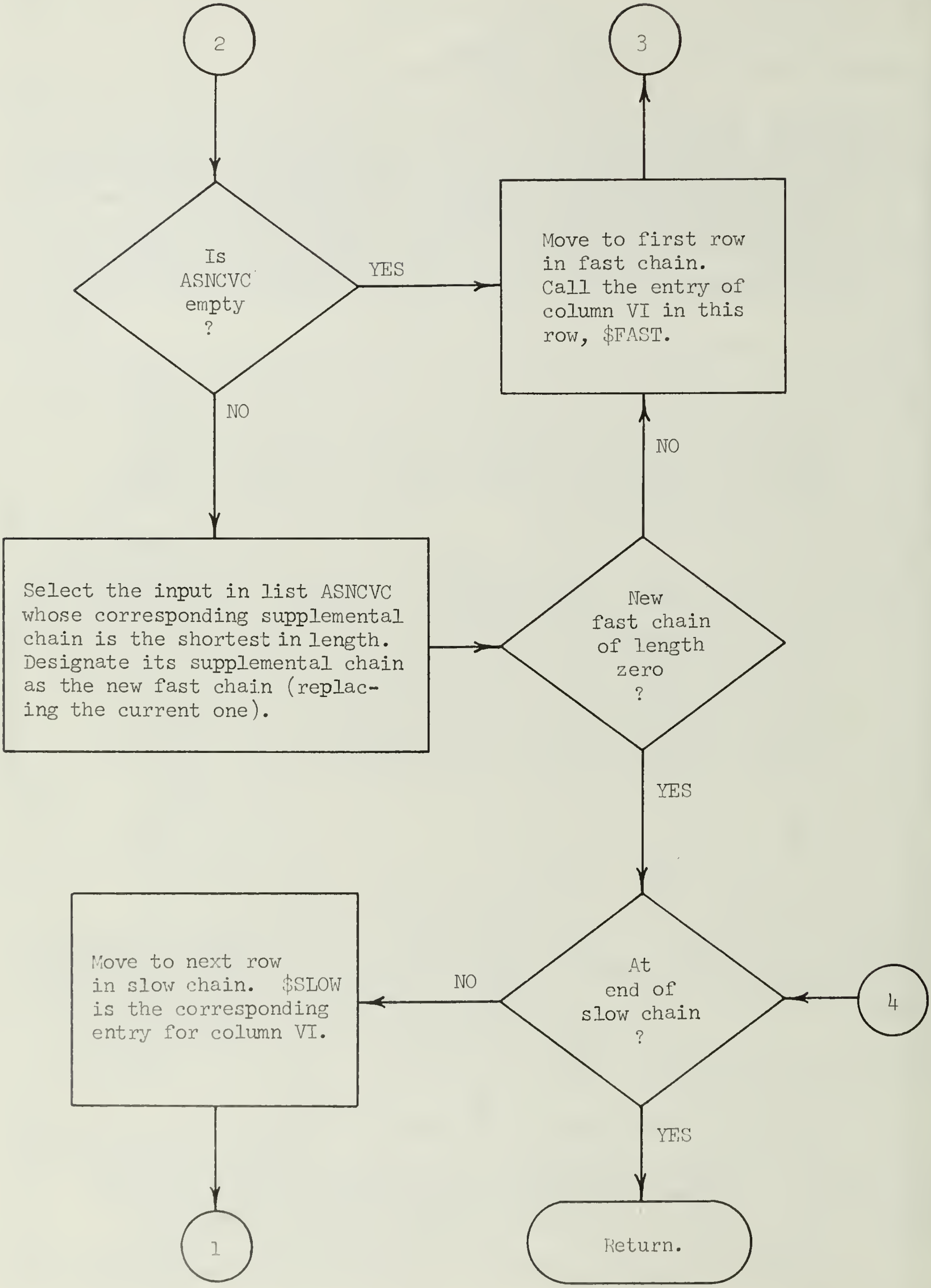

2
3
Is ASNCVC empty ?
YES
Move to first row in fast chain. Call the entry of column VI in this row, $FAST.
NO
NO
Select the input in list ASNCVC whose corresponding supplemental chain is the shortest in length. Designate its supplemental chain as the new fast chain (replacing the current one).
New fast chain of length zero ?
YES
Move to next row in slow chain. $SLOW is the corresponding entry for column VI.
NO
At end of slow chain ?
4
YES
1
Return.

Fig. 6.2.1.7 (Continued)

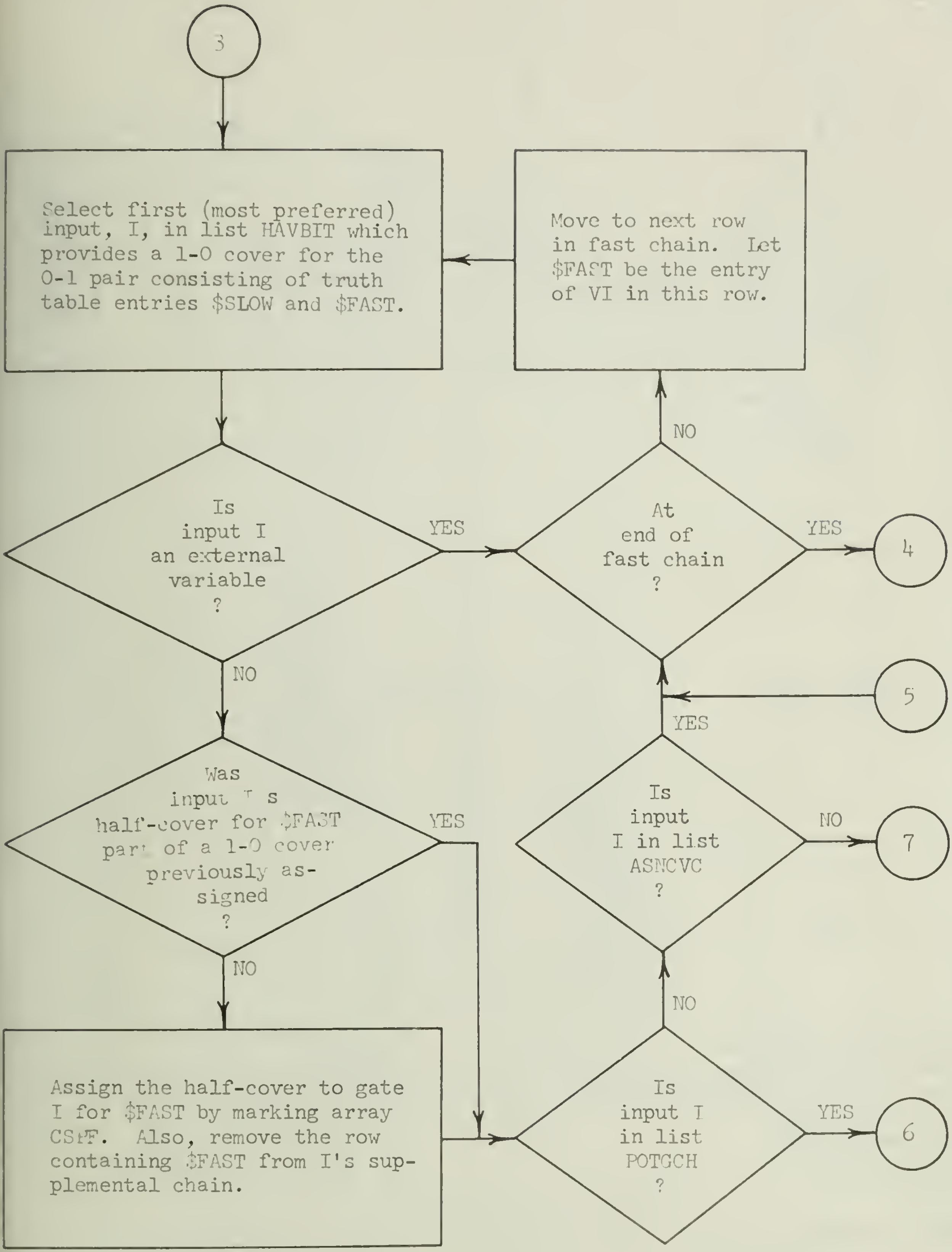

3
Select first (most preferred) input, I, in list HAVBIT which provides a 1-0 cover for the 0-1 pair consisting of truth table entries $SLOW and $FAST.
Move to next row in fast chain. Let $FAST be the entry of VI in this row.
NO
Is input I an external variable ?
YES
At end of fast chain ?
YES
4
NO
5
YES
Was input I s half-cover for $FAST part of a 1-0 cover previously as-signed ?
YES
Is input I in list ASNCVC ?
NO
7
NO
NO
Assign the half-cover to gate I for $FAST by marking array CSPF. Also, remove the row containing $FAST from I's sup-plemental chain.
Is input I in list POTGCH ?
YES
6

Fig. 6.2.1.7 (Continued)

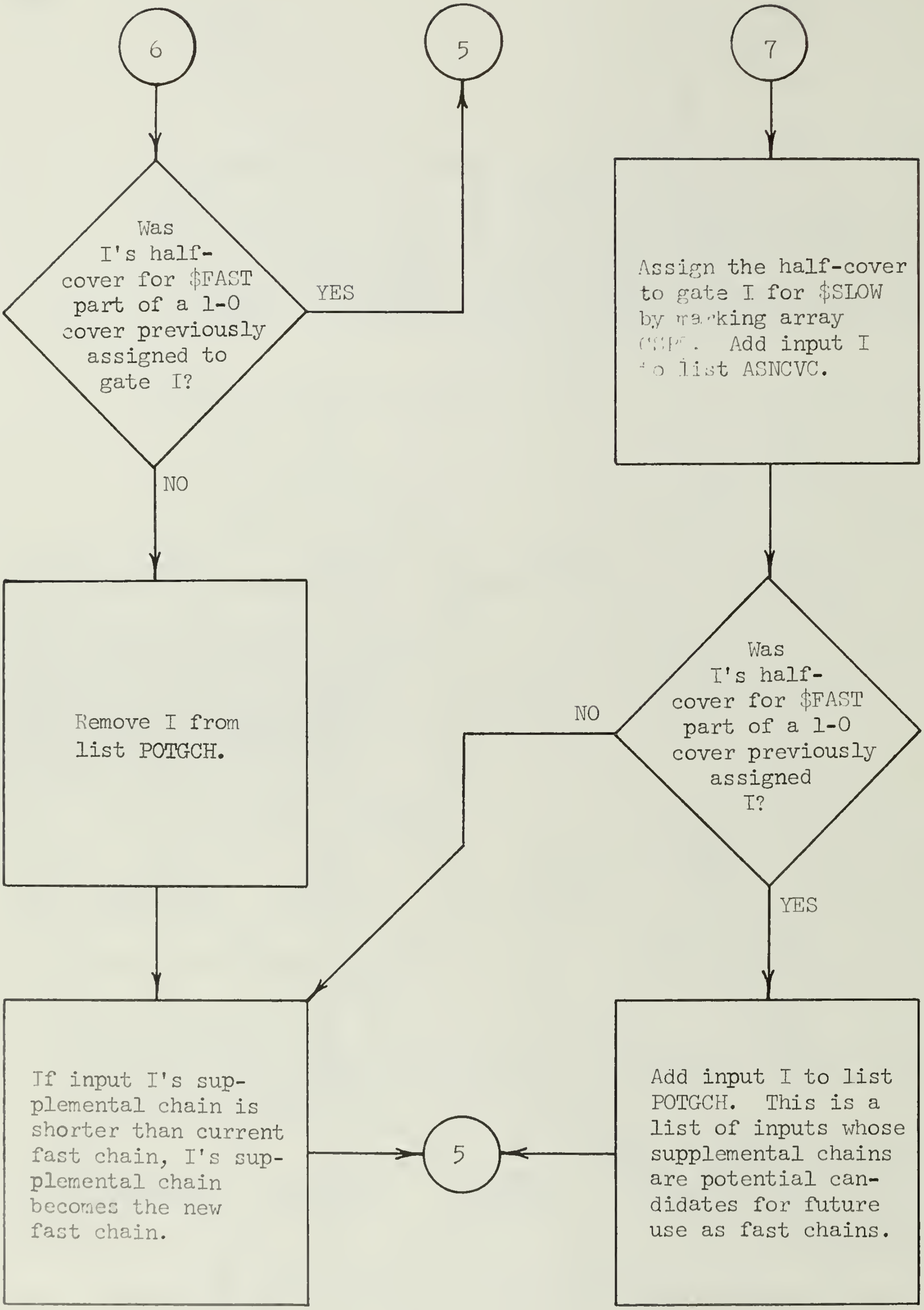

6
5
7
Was I's half-cover for $FAST part of a 1-0 cover previously assigned to gate I?
YES
NO
Assign the half-cover to gate I for $SLOW by marking array CSPF. Add input I to list ASNCVC.
Remove I from list POTGCH.
Was I's half-cover for $FAST part of a 1-0 cover previously assigned I?
NO
YES
If input I's supplemental chain is shorter than current fast chain, I's supplemental chain becomes the new fast chain.
5
Add input I to list POTGCH. This is a list of inputs whose supplemental chains are potential candidates for future use as fast chains.

Fig. 6.2.1.7 (Continued)

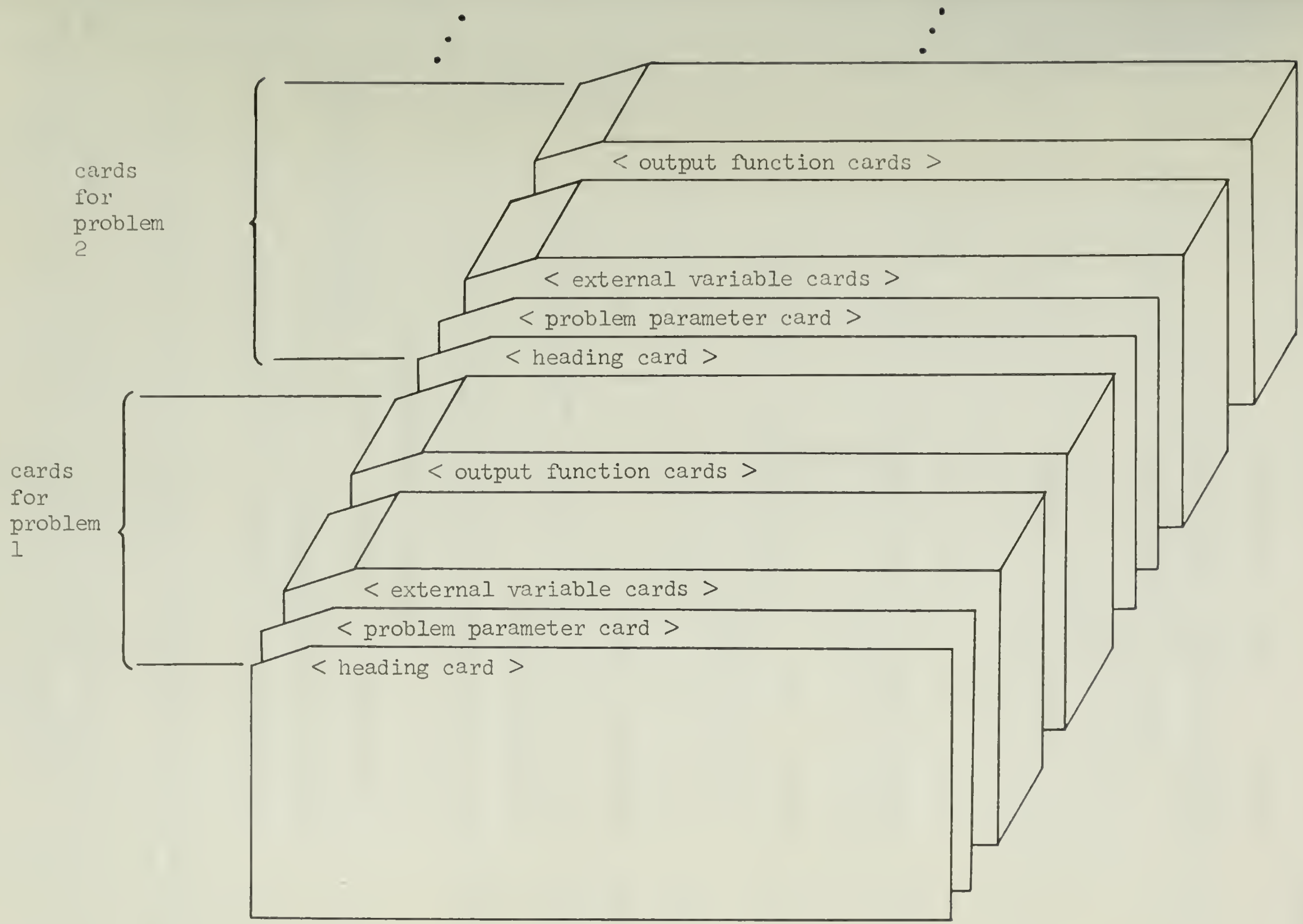

cards
for
problem
2
< output function cards >
< external variable cards >
< problem parameter card >
< heading card >
cards
for
problem
1
< output function cards >
< external variable cards >
< problem parameter card >
< heading card >

Fig. 6.2.2.1 Arrangement of data cards for submission of transduction problems to program MOSTRA.

Columns

1 - 4 Number of external variables, n.

5 - 8 Number of network output functions, m.

9 - 12 'U' indicates that the output functions will be expressed in explicit form;[†]

'X' indicates that the output functions will be expressed in implicit form.[‡]

Default (i.e., a blank field) is implicit form.

13 - 16 Maximum number of FET's permitted in series.[††]

Default is 10,000.

17 - 20 Maximum number of FET's permitted in parallel.[††]

Default is 10,000.

21 - 24 Maximum number of levels permitted in a network.[††]

Default is 10,000.

25 - 28 Maximum permitted fan-out of a negative gate.[††]

Default is 10,000.

29 - 32 Value indicating preference ordering to be used in the assignment of 1-0 covers (ordering σ_3):

'1' indicates the ordering: external variables, high level gates, low level gates;

'2' indicates the ordering: external variables, low level

[†] In the explicit representation, output functions are defined only for explicitly given input vectors. (This will be further discussed later.)

[‡] In the implicit representation, output functions must be defined (0,1, or *) for all 2^n input vectors. The correspondence between input vectors and output function values is implicit in the order in which output function values are given. (This will be further discussed later.)

[††] These specifications are currently not used by the program. They are intended to be utilized in future expansions of the program's capabilities.

gates, high level gates.

Default is 2.

33 - 36 Value indicating whether or not cell configurations are to be determined when negative completions are made:

'1' indicates only negative completions are to be made;

'2' indicates cell configurations are to be determined while negative completions are made.

Default is 2.

37 - 40 Value indicating whether pruning or general transduction procedure is to be applied:

'1' indicates pruning procedure;

'2' indicates general transduction procedure.

Default is '2'.

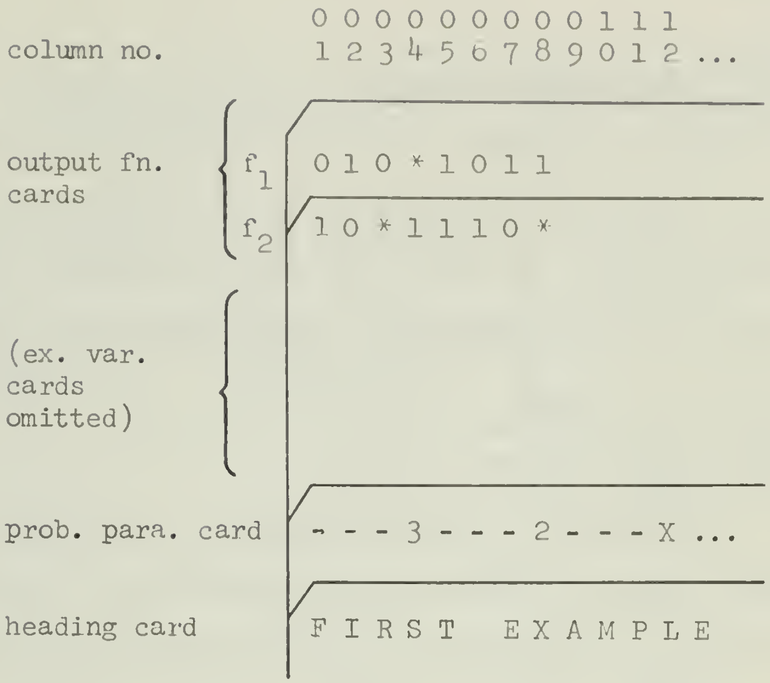
The program checks whether: $2 \leq n \leq 10$; $1 \leq m \leq 10$; columns 9-12 contain '_ _ _ U' ('_' represents a blank column), '_ _ _ X', or '_ _ _ _'; numbers in columns 13-16, 17-20, 21-24, and 25-28 are all non-negative. If any of these conditions are violated, an error message is printed and the problem is flushed if possible (otherwise, the program terminates).

External variable cards (Omitted if network output functions are expressed in implicit form.) In the explicit representation, the values of the network output functions can be specified for only up to 512 input vectors (some input vectors may be incompletely specified). On each external variable card only columns 1-64 are reserved to express components of input vectors, and anything may be contained in columns 66-80 (column 65 is also reserved) which are ignored by the program. Columns 1-64 may contain only the characters '*', '0', '1', or '/'. Any other character will cause an error message to be printed and the problem to be abandoned. Suppose ℓ is the number of input vectors to be explicitly expressed. For each external variable x_i, $i = 1,\ldots,n$, $\lceil \ell/64 \rceil$ cards are used to specify the external variable: $x_i^{(d)}$ is given in column $(d - \lfloor (d - 1)/64 \rfloor)$ of the $(\lceil d/64 \rceil)$th card $d = 1,\ldots,\ell$. A slash, '/', must be placed immediately after the last 0, 1, or * for external variable x_1 (even if it must be placed in column 65). If no such delimiter is encountered within the first 8 cards read, an error message is printed and the program terminates. External variable cards can be omitted only when output function values are expressed for all 2^n different input vectors on the output function cards (such a case is referred to as the implicit representation since the correspondence of input vector to output function value is implicit).

Output function card(s) On each output function card, only columns 1-64 are reserved to express components of vectors representing output functions. Columns 65-80 are ignored by the program and may contain any information. Any characters other than '0', '1', or '*' will cause an error message to be printed and the problem to be flushed. Suppose ℓ is the number of input vectors, explicitly ($\ell \leq 512$, $\ell < 2^n$) or implicitly ($\ell = 2^n$) expressed, for which output functions are to be given. For each output function f_i, $i = 1,\ldots,m$, $\lceil \ell/64 \rceil$ cards are used to specify the output function: $f_i^{(d)}$ is given in column $(d - \lfloor (d - 1)/64 \rfloor)$ of the $(\lceil d/64 \rceil)$th card, $d = 1,\ldots,\ell$. In the explicit representation, it is possible for the user to acidentaly give an inconsistent specification of the desired output functions. All such inconsistencies will be detected; an error message will be printed; and the problem will be abandoned.

Fig. 6.2.2.2 gives examples of two sets of data cards for two different transduction problems. In (a), the required output functions are expressed on the data cards in implicit form. In (b), they are expressed in explicit form.

x_1	x_2	x_3	f_1	f_2
0	0	0	1	0
0	0	1	0	1
0	1	0	*	0
0	1	1	1	*
1	0	0	1	1
1	0	1	1	0
1	1	0	0	1
1	1	1	*	1



column no. 000000000111
123456789012 ...

output fn. cards f_1 010*1011

f_2 10*1110*

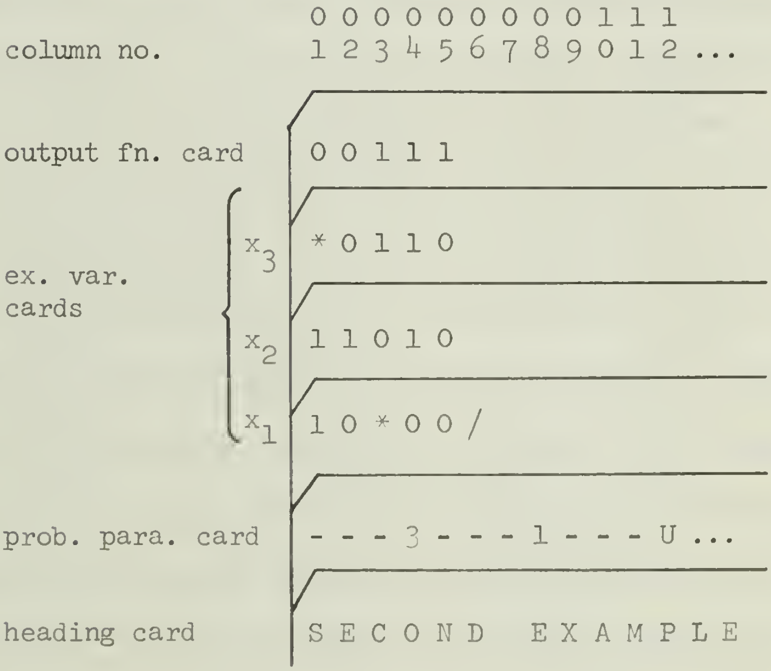
(ex. var. cards omitted)

prob. para. card - - - 3 - - - 2 - - - X ...

heading card FIRST EXAMPLE

(a) Example, implicit specification of output functions.

x_1	x_2	x_3	f_1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	*
1	0	1	1
1	1	0	0
1	1	1	0



column no. 000000000111
123456789012 ...

output fn. card 00111

ex. var. cards x_3 *0110

x_2 11010

x_1 10*00/

prob. para. card - - - 3 - - - 1 - - - U ...

heading card SECOND EXAMPLE

(b) Example, explicit specification of output functions.

Fig. 6.2.2.2 Examples of sets of input data cards.

6.2.3 Experimental results

Since the purpose here is to demonstrate the feasibility of the transduction approach rather than to solve any specific body of problems, functions were arbitrarily selected for 29 test problems: ten problems each involving 4- and 5-variable functions, three problems each involving 6-, 7-, and 8-variable functions. The set of test problems involve the synthesis of 1-, 2-, and 3-output MOS cell networks.

The results of these 29 experiments are shown in Table 6.2.3.1. Each problem was run 4 times with program MOSTRA: both the pruning procedure and the general procedure were executed with both available σ_3 orderings. Each of these four combinations was repetitively applied in accordance with the flowchart of TESTSQ in Fig. 6.2.1.3 (i.e., each transduction procedure with choice of σ_3 was applied until the network cost failed to decrease, or for a minimum of two applications – whichever was smaller). All transduction procedures started from the same initial network synthesized by the program using the n-cube labelling method of Nakamura et al.[NTK 72] Since the number of cells in these initial networks is already minimized (for multiple-output cases, the number of gates is minimal only under the selected pre-specification of output cell positions), MOSTRA attempts to reduce the number of connections over which the synthesis algorithm of [NTK 72] has no control. The results obtained by the general transduction procedure were omitted from the table since they were almost identical to the results obtained by the corresponding pruning procedure (different results were obtained for only two problems for which the general transduction procedure achieved slightly worse results – more suitable problems for the general transduction procedure will be discussed shortly). The network 'costs'

Problem number	Number of ex. vars.	Number of output fns.	Cost of initial network†	Pruning Procedure, MOSTRA		DIMN of [Yam 76]
				Best cost with cover preference:† 1)ex. vars. 2)high level cells 3)low level cells (ORDERING 1)	Best cost with cover preference:† 1)ex. vars. 2)low level cells 3)high level cells (ORDERING 2)	Cost of synthesized network†
1	4	1	3/13/15	3/11/13	3/11/13	3/9/12
2	4	1	3/13/23	3/11/20	3/12/20	3/12/19
3	4	1	2/9/21	2/9/21	2/9/21	2/9/21
4	4	1	2/6/7	2/5/6	2/5/6	2/5/6
5	4	2	4/20/27	4/20/27	4/18/27	4/16/27
6	4	2	4/20/43	4/18/36	4/18/36	4/21/32
7	4	2	4/21/44	4/20/35	4/18/31	4/18/31
8	4	3	5/29/59	5/29/56	5/29/56	5/28/60
9	4	3	5/27/51	5/24/36	5/24/36	5/19/28
10	4	3	5/28/56	5/28/51	5/28/51	5/29/53
11	5	1	3/18/33	3/16/26	3/16/24	3/16/29
12	5	1	3/17/36	3/16/33	3/16/33	3/15/33
13	5	1	3/18/50	3/14/39	3/15/44	3/15/31
14	5	1	3/17/45	3/16/44	3/16/44	3/15/45
15	5	2	4/25/61	4/25/61	4/25/61	4/25/61

†'i/j/k' denotes i cells, j intercell connections, and k driver FET's.

Table 6.2.3.1 Results by program MOSTRA for 29 MOS network synthesis problems. (Resulty by program DIMN[Yam 76] are also included.)

Problem number	Number of ex. vars.	Number of output fns.	Cost of initial network	Pruning Procedure, MOSTRA		DIMN of [Yam 76]
				Best cost with cover preference:† 1)ex. vars. 2)high level cells 3)low level cells (ORDERING 1)	Best cost with cover preference: 1)ex. vars. 2)low level cells 3)high level cells (ORDERING 2)	Cost of synthesized network
16	5	2	4/26/88	4/26/76	4/26/82	4/26/83
17	5	2	4/26/84	4/26/78	4/26/78	4/26/66
18	5	3	6/42/145	6/38/142	6/40/143	6/38/102
19	5	3	5/35/117	5/35/117	5/35/117	5/35/117
20	5	3	6/43/160	6/37/114	6/38/125	6/39/90
21	6	1	3/21/83	3/20/80	3/20/80	3/20/88
22	6	2	5/39/260	5/33/237	5/35/236	5/35/204
				5/36/236‡		
23	6	3	6/49/279	6/43/279	6/45/281	6/43/234
24	7	1	4/34/234	4/30/201	4/30/195	4/25/176
25	7	2	5/45/503	5/44/465	5/44/459	5/44/408
26	7	3	6/57/738	6/56/707	6/56/691	6/54/607
27	8	1	4/38/497	4/35/462	4/36/462	4/33/410
28	8	2	5/50/1174	5/50/1127	5/50/1115	5/50/921
29	8	3	6/63/1570	6/63/1481	6/63/1473	6/63/1336

‡This solution does not have the 'best cost'; however, it has the least number of FET's among those solutions obtained.

Table 6.2.3.1 (Continued)

given are the lowest attained during the repetitive application of each transduction procedure for each synthesis problem. In each case except the one noted in Table 6.2.3.1 (the pruning procedure with ordering 1 for problem 22) a solution with fewest connections also had the fewest FET's. It should be noted that the cell configurations derived are 'unfactored.' Factoring can reduce the numbers of FET's. To provide some basis for comparison, the results of program DIMN[Yam 76] are also included. For six of the test problems the pruning procedure, with one or both of the two different orderings, achieved results of lower costs than did DIMN, while DIMN obtained results of lower costs in nine instances. Any comparison, however, of the results by MOSTRA and those by DIMN should recognize the fact that MOSTRA is starting with an initial network which is synthesized by a method which does not attempt to minimize the number of intercell connections or MOSFET's used. It is quite possible that the use of a different type of initial network would improve the results obtained by MOSTRA.

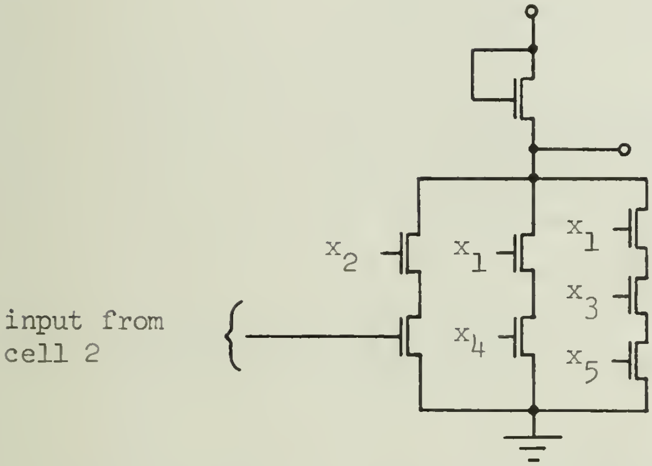
The pruning procedure with ordering 1 (cover preference: external variables, high level cells, low level cells) reduced the number of connections of the initial network in 66% of the cases. With ordering 2 (cover preference: external variables, low level cells, high level cells), the numbers of connections were reduced in 69% of the cases. For both orderings, the number of FET's was reduced in 86% of the cases. The largest percentage decrease in the number of connections was 22% for problem 13 (ordering 1). Among problems in which reductions in the numbers of connections were achieved, the pruning procedure with ordering 1 averaged a 10% decrease while the same procedure with ordering 2 averaged a 9% decrease. Slghtly better improvements were obtained with respect to the numbers of FET's Among problems in which reductions in the numbers of FET's were achieved, the pruning procedure with both ordering 1 and ordering 2 averaged an 11% decrease in numbers of FET's. The largest percentage decreases were 30% for problem 7 (ordering 2), and 29% for problems 9 (both orderings) and 20 (ordering 1).

For the majority of the problems, the pruning procedure was applied (iteratively) only twice by MOSTRA after synthesizing the initial network. For ordering 1, the pruning procedure was applied three times for nine of the problems. In the case of ordering 2, three applications of the pruning procedure were used for only five of the 29 problems.

Two examples are given, in Figs. 6.2.3.2 and 6.2.3.3, in which the initial networks are compared with the best results achieved by applying the pruning procedure. (The networks are shown as they appear in the actual printout. Only the driver sections of the MOS cells are shown. The notation 'UI' in the figures denotes an FET controlled by the output of cell I, while 'XI' denotes an FET controlled by x_I. Fig. 6.2.3.1 shows an example of this straightforward correspondence between the computer printout of a driver's configuration and the constructed MOS cell.) Fig. 6.2.3.2(a) shows the initial network synthesized by MOSTRA for problem 13. The network consists of 3 cells, 18 intercell connections, and 50 driver FET's. After the second application of the pruning procedure (ordering 1) starting from this initial network, the network in Fig. 6.2.3.2(b) is obtained. This network also consists of 3 cells, but contains only 14 intercell connections and 39 driver FET's. The initial network synthesized by MOSTRA for problem 20 is shown in Fig. 6.2.3.3(a). It consists of 6 cells, 43 intercell connections, and 160 driver FET's. After two applications of the pruning procedure (ordering 1), the result shown in Fig. 6.2.3.3(b)

```
   |--X1--X3--X5--|
0--|--X1--X4------|--0
   |--X2--U2------|
```

(a) Driver configuration as appearing in computer printout.



(b) MOS cell corresponding to (a).

Fig. 6.2.3.1 Example of correspondence between computer printout of a driver's configuration and the constructed MOS cell.

```
          CELL:      LEVEL     FED  BY:
            1        / 1/        X1   X2   X3   X4   X5   2   3
            2        / 2/        X1   X2   X3   X4   X5   3
            3        / 3/        X1   X2   X3   X4   X5

     **   NETWORK  COST =      3018



                        |--X1--X3--X4--X5--|
                        |--X1--X2--X3--X5--|
                        |--X1--X2--X3--X4--|
                        |--X4--X5--U2------|
          CELL  1  C--  |--X2--X4--U2------|--O
                        |--X1--X5--U2------|
                        |--X1--X4--U2------|
                        |--X1--X2--U2------|
                        |--X4--U3----------|
                        |--X2--U3----------|


                        |--X1--X4--X5--|
                        |--X1--X2--X5--|
                        |--X1--X2--X4--|
          CELL  2  O--  |--X2--X3------|--O
                        |--X1--X3------|
                        |--U3----------|


                        |--X2--X4--|
          CELL  3  C--  |--X5------|--O
                        |--X3------|
                        |--X1------|


     TOTAL  NUMBER  OF  DRIVER  FET'S  IN  NETWORK  IS  50.
```

(a) Initial network synthesized by MOSTRA.

Fig. 6.2.3.2 Example of pruning results.

```
        CELL:      LEVEL    FED  BY:
          1        / 1/      X1   X2   X3   X4   X5    2    3
          2        / 2/      X1   X2   X3   X4   X5
          3        / 2/      X3   X5

   **  NETWORK  COST  =    3014



                      |--X1--X3--X4--X5--|
                      |--X1--X2--X3--X5--|
                      |--X1--X2--X3--X4--|
        CELL  1  0--  |--X4--U2--U3------|--0
                      |--X4--X5--U2------|
                      |--X2--U2--U3------|
                      |--X1--X5--U2------|


                      |--X1--X4--X5--|
                      |--X1--X2--X5--|
        CELL  2  0--  |--X1--X2--X4--|--0
                      |--X2--X3------|
                      |--X1--X3------|


                      |--X5--|
        CELL  3  0--  |--X3--|--0


   TOTAL  NUMBER  OF  DRIVER  FET'S  IN  NETWORK  IS  39.
```

(b) Network derived by two applications of pruning procedure (with ordering 1).

Fig. 6.2.3.2 (Continued)

```
CELL      LEVEL      FED   BY:

  1       / 1/       X1    X2   X3   X4   X5   2   3   4   5
  2       / 2/       X1    X2   X3   X4   X5   3   4   5   6
  3       / 3/       X1    X2   X3   X4   X5   4   5
  4       / 4/       X1    X2   X3   X4   X5   5   6
  5       / 5/       X1    X2   X3   X4   X5   6
  6       / 6/       X1    X2   X3   X4   X5

**  NETWORK  COST  =    6043
```

```
             |--X1--X3--X4--X5--|
             |--X1--X2--X3--U5--|
             |--X5--U2--U4------|
             |--X4--U3--U4------|
             |--X4--X5--U5------|
CELL  1  0-- |--X4--X5--U4------| --0
             |--X3--U4--U5------|
             |--X3--X5--U5------|
             |--X3--X4--U5------|
             |--X2--X5--U5------|
             |--X2--X4--U5------|
```

```
             |--X5--U3--U4--U5--|
             |--X3--X4--U3--U4--|
             |--X1--U3--U4--U5--|
             |--X1--X3--X5--U5--|
             |--X1--X2--X5--U4--|
CELL  2  0-- |--X4--X5--U4------| --0
             |--X3--U4--U5------|
             |--X3--X4--U5------|
             |--X2--U4--U5------|
             |--X1--X2--U5------|
             |--U6--------------|
```

```
             |--X2--X4--X5--U4--|
             |--X1--X5--U4--U5--|
             |--X1--X4--X5--U4--|
CELL  3  0-- |--X1--X3--X4--U5--| --0
             |--X1--X3--X4--U4--|
             |--X4--X5--U5------|
             |--X2--X5--U5------|
             |--X2--X4--U5------|
```

```
             |--X2--X3--X4--X5--|
             |--X1--X2--X4--X5--|
             |--X4--X5--U5------|
             |--X3--X5--U5------|
             |--X3--X4--U5------|
CELL  4  0-- |--X2--X5--U5------| --0
             |--X2--X4--U5------|
             |--X2--X3--U5------|
             |--X1--X4--U5------|
             |--X1--X3--U5------|
             |--X1--X2--U5------|
             |--U6--------------|
```

Fig. 6.2.3.3   Example of pruning results.

```
                 |--X3--X4--X5--|
                 |--X2--X4--X5--|
                 |--X2--X3--X5--|
CELL  5   0--|--X2--X3--X4--|--0
                 |--X1--X4--X5--|
                 |--X1--X2--X4--|
                 |--U6----------|

                 |--X5--|
                 |--X4--|
CELL  6   0--|--X3--|--0
                 |--X2--|
                 |--X1--|

TOTAL  NUMBER  OF  DRIVER  FET'S  IN  NETWORK  IS  160.
```

(a) Initial network synthesized by MOSTRA.

Fig. 6.2.3.3 (Continued)

```
CELL:     LEVEL     FED  BY:

  1        / 1/      X1    X2    X3    X4    X5    2    3    4    5
  2        / 2/      X1    X2    X3    X4    X5    3    4    5    6
  3        / 3/      X1    X2    X3    X4    X5    4    5
  4        / 4/      X1    X2    X3    X4    X5    5
  5        / 5/      X1    X2    X3    X4    X5
  6        / 3/      X4

NETWORK  COST  =   6037
```

```
              |--X1--X3--X4--X5--|
              |--X1--X2--X3--U5--|
              |--X5--U2--U4------|
              |--X4--U4--U5------|
CELL  1  0--  |--X4--U3--U4------|  --0
              |--X4--X5--U4------|
              |--X3--U4--U5------|
              |--X3--X5--U5------|
              |--X3--X4--U5------|
              |--X2--X5--U5------|

              |--U3--U4--U5--U6--|
              |--X3--X4--U3--U4--|
              |--X2--X4--X5--U5--|
CELL  2  0--  |--X1--X4--X5--U4--|  --0
              |--X1--X3--X5--U5--|
              |--X3--X4--U5------|
              |--X1--X2--U6------|

              |--X1--X5--U4--U5--|
              |--X1--X4--X5--U4--|
              |--X1--X3--X4--U5--|
CELL  3  0--  |--X1--X3--X4--U4--|  --0
              |--X4--X5--U5------|
              |--X2--X5--U5------|
              |--X2--X4--U5------|

              |--X3--X5--U5--|
              |--X2--X4--X5--|
CELL  4  0--  |--X2--X3--U5--|  --0
              |--X1--X4--U5--|
              |--X1--X3--U5--|

              |--X3--X4--X5--|
              |--X2--X3--X5--|
CELL  5  0--  |--X2--X3--X4--|  --0
              |--X1--X4--X5--|
              |--X1--X2--X4--|

CELL  6  0-----X4-----0


TOTAL  NUMBER  OF  DRIVER  FET'S  IN  NETWORK  IS  114.
```

(b) Network derived by two applications of pruning procedure (ordering 1).

Fig. 6.2.3.3 (Continued)

is obtained: a network of 6 cells, 37 intercell connections, and only 114 driver FET's.

Noting the failure of the general transduction procedure to improve upon the results of the pruning procedure for the 29 test problems, it was conjectured that the general transduction procedure might be best applied in cases in which a relatively large number of network output functions are required. Such networks seemed to offer more 'flexibility' for the exchange of input connections performed by the general transduction procedure. 28 more test problems were chosen involving arbitrarily selected 4- and 5-variable functions and 3-, 4-, 5-, and 6-output networks. The MOSTRA program was run with these problems in an effort to find problems in which the ability of the general transduction procedure to add new connections was utilized. In the solution of 61% of these problems, new connections were found to have been added to the network. Of these, five problems showing the largest reductions in numbers of connections and FET's were rerun by MOSTRA using the pruning procedure (with ordering 2).

From initial networks of costs 7/38/85, 8/47/99, 9/54/98, 9/54/108, and 6/29/48, the general transduction procedure obtained networks of costs 7/37/64, 8/46/80, 9/48/81, 9/51/93, and 6/25/37, respectively, while the pruning procedure obtained networks of costs 7/38/85, 8/47/81, 9/49/87, 9/52/84, and 6/27/41, respectively.

6.3 Discussion of Transduction Approach

Judging from the results of the previous section, the transduction approach can obviously be utilized to improve networks synthesized by certain other methods, and the combination of a simple synthesis method and the transduction procedures can itself be considered a synthesis method.

Compared with the methods of [Lai 76] (programmed by Yamamoto[Yam 76] which produce irredundant MOS networks, the transduction approach offers flexibility. Two apparent drawbacks of the transduction approach, however, are: (1) the transduction approach generally requires several iterations (i.e., applications of the transduction procedures) and, hence, generally more computation time and (2) at least for the initial network synthesis algorithm used in the preceding examples, the results of the transduction approach seemed slightly inferior in many cases.

The former problem could be minimized by not deriving MOS cell configurations until after the last application of a transduction procedure (this derivation is generally much more time-consuming than the transduction procedure itself - often by a factor of more than ten). The user is not currently offered such a choice.

The latter problem might be lessened by the identification of a more suitable algorithm for initial network synthesis in MOSTRA and/or the implementation of modifications discussed in Section 6.1 which are not currently included in the program. Also, new transduction procedures can be created, and different transduction procedures can be used together for improved results. (In the case of NOR gates, the 'error-compensation' class of transduction procedures [KLCM 75][LC 75] were the most powerful found. They were also the most complex. Comparable transduction procedures for MOS networks might also be more powerful than the pruning or general transduction procedures. The creation of 'error-compensation' -type transduction procedures for MOS networks is not considered in this work, however, due to the lack of sufficient available research time.)

While the performance of the general transduction procedure was less than anticipated (theoretically, it could have reduced the numbers of cells in some of the multiple-output networks of the test problems, although this did not occur), it does appear useful for networks having several outputs. Also, improvements in the general transduction procedure itself (e.g., more sophisticated methods of deciding which new connections to add and which old connections to retain) or its implementation as part of a computer program (e.g., as discussed in the description of subroutine GENER1 in Section 6.2.1, certain new connections which are possible under Algorithm 6.1.3 (GP) were not permitted by the subroutine) could lead to better results.

The greatest potential for the transduction approach, however, is not in competition with G-minimal synthesis methods such as those of [Lai 76], but in another area. As can be seen by the test results in the preceding section, cells in G-minimal networks rapidly grow too large for practical implementation as the number of external variables increases. The transduction approach obviously need not use G-minimal networks as starting points (i.e., as initial networks). Thus, for example, the pruning procedure can be applied to a network of practical sized cells (possibly obtained by hand or by a procedure — perhaps employing the transformations of Section 5 — programmed for this purpose), and the result can generally be anticipated to be a reduced network still consisting of cells of a practical size. Another possible application of the general transduction procedure is in the combination of two or more separately designed networks into a single network of lower cost. In the case of the general procedure, however, there is a somewhat greater risk that the result, after starting with an initial network satisfying practical limitations, may be impractical to implement. In the event of a failure, any impractical networks can always be discarded in favor of the initial network.

It is hoped that the current state of the MOSTRA program will be improved with: the capability to accept pre-designed initial networks; the capability to generate initial networks by several different algorithms selectable by the user; the addition of new transduction procedures (possibly some considering maximum fan-out, maximum FET's in series, the division of overly complex cells, etc.); the improvement of existing transduction procedures; and the capability of the user to control the types and sequences of transduction procedures to be applied to a given problem.

It may be of interest to make some final comments on the success of the transduction approach for NOR network synthesis (see [Cul 75], [CLK 74], [KC 76], [KM 76], [KLM tbp], [KLCM 75], [Lai 75], [LC 74], [LC 75], [LK 75]) versus the results obtained thus far of the transduction approach to negative gate (MOS cell) network synthesis.

In the NOR case, initial networks chosen for the transduction procedures generally contain a significant percentage of 'redundant' gates (i.e., a number of gates in excess of the minimum number required to realize the given function or functions). This is due to the relatively large amount of computational effort required to obtain an initial network with an optimal or near-optimal number of gates (in fact, one of the reasons to use the transduction approach is to avoid this very type of computational effort). On the other hand, it is relatively easy to obtain a negative gate network having the minumum number of gates required to realize a given function (in the multiple-output case, it is difficult to obtain a network of the minimum number of gates, but networks having a near-optimal number are obtained as easily as for the single-output case). If such initial networks are used (as in the preceding experiments), the transduction procedures for negative gate networks obviously have little or no possibility to remove gates and fewer chances to 're-configure' (i.e., change the interconnection pattern among external variables and gates) networks than in the NOR gate case. This, in turn, means fewer opportunities to pursue in the search for solutions of lower cost. In the case of NOR networks, many of the redundant gates in the initial networks can be easily removed by transduction procedures.

Perhaps a more significant consideration is the greater 'interdependency' among negative gates (as opposed to NOR gates) caused by the required 1-0 covers. As previously mentioned, the number of covers required for a negative gate is generally much greater than that for a NOR gate in networks with the same number of external variables. This, coupled with the fact that the minimum number of negative gates required to realize a given function (or set of functions) grows very slowly (proportional to $\log_2$) with an increase in the number of external variables, makes the removal of connections from negative gate networks with optimal or near-optimal numbers of gates exceedingly difficult, if not impossible, as the number of external variables grows. In the case of NOR networks, many redundant connections can exist even in a network of an optimal or near-optimal number of gates. In other words, if GI-minimal negative gate and NOR gate networks having the same number of gates were compared, the negative gate networks would, on the average, contain many more connections. It is conjectured that in many of these cases, the negative gate networks for a given function or functions may require the maximum possible numbers of connections (i.e., no connections can be deleted from the generalized form of a feedforward network of negative gates in Fig. 2.11).

For these reasons, transduction approach improvements of network costs in the negative gate case can not be expected to be of the same magnitude as those in the NOR gate case. Furthermore, applying transduction procedures for negative gate networks can be anticipated to require significantly more computational effort.

While established transduction procedures for NOR gates can serve as guidelines for the development of corresponding types of transduction procedures for negative gates, the differing characteristics of negative gates (e.g., the stronger covering requirement, the 'non-fixed' function of a negative gate, the 'connectability' of any external variable or gate not causing a loop in the network (NOR gates have more restrictive conditions for connectability), etc.) will require much more than a straightforward adaptation of the transduction procedures for NOR gates.

7. IRREDUNDANT NETWORKS AND TEST SET GENERATION

[Lai 76] gives an algorithm which can synthesize G-minimal, single-output, irredundant networks of MOS cells for given completely or incompletely specified functions (Algorithm 8.6 in [Lai 76]). The term 'irredundant' means that no FET or group of FET's can be extracted (i.e., replaced by a short- or open-circuit) from the network without changing the desired (specified) output of the network. A network which is irredundant is a 'diagnosable network.'

Diagnosable networks of MOS cells are also treated in [Pai 73]. The networks synthesized by the algorithm of [Pai 73] are, however, 2-level, non-G-minimal[†] networks, and require the accessibility of auxiliary test points within the network.

The networks synthesized by the algorithm of [Lai 76] require no extra terminals for the administration of tests to detect faulty FET's (i.e., only network inputs and outputs need be accessible). Sets of input vectors are proposed in [Lai 76] which constitute 'test sets' for the detection of certain classes of faults. These test sets are quite large, however (the test for all possible faults being V_n itself), due, at least in part, to their 'universal' nature.

This section (Section 7) will propose a method for obtaining smaller test sets 'tailored' to a particular irredundant network synthesized by the algorithm of [Lai 76]. First, a background must be established for the presentation of the algorithm of [Lai 76] and the algorithm for test set generation. The following notation and definitions are adopted from [Lai 76]:

[†] Since the algorithm requires a number of second level cells equal to the number of prime implicants in an irredundant disjunctive form expressing the desired function, there can be up to 2^{n-1} cells in the second level — substantially more than the maximum of $\lfloor(n + 1)/2\rfloor$ cells in the second level of a 2-level, G-minimal network.

An incompletely specified function will be denoted $\tilde{f}$. A completion of $\tilde{f}$ will be denoted f. A completely specified function can be considered a special case of an incompletely specified function, i.e., it is possible that $\tilde{f} \equiv f$.

Definition 7.1 An FET is said to be in the stuck-at-short failure mode (or stuck-at-short fault) if it becomes permanently conductive.

Definition 7.2 An FET is said to be in the stuck-at-open failure mode (or stuck-at-open fault) if it becomes permanently nonconductive.

The justification of this fault model is supported by an engineering analysis in [SK 69] as discussed in [Pai 73].

Definition 7.3 An MOS network is said to be diagnosable if and only if every individual or set of stuck-at-short and/or stuck-at-open faults in the network (referred to as single fault and multiple fault, respectively) can be detected by a comparison of the network's actual output with the required (error-free) output function.

Note that this definition of 'diagnosability' only requires the detection of faults and not the location of faulty FET. It is clear that a network is irredundant if and only if it is diagnosable.

Let N denote an MOS network consisting of k driver FET's, $D_1,\ldots,D_k$, and realizing a function f(N) (the actual output of a network is always completely specified in terms of its inputs) of n variables. Network N with one or more faulty FET's is denoted by N(F) where F is the set of faulty FET's with their respective failure modes expressed as D_i^s if D_i is stuck-at-short and D_i^σ if D_i is stuck-at-open. f(N(F)) denotes the output function of N(F).

Also, let $f(N,A)$ and $f(N(F),A)$ denote the outputs of networks N and N(F), respectively, for input vector $A \in V_n$.

<u>Definition 7.4</u> A <u>test</u> for a faulty network N(F) is an input vector $A \in V_n$ for which $f(N,A) \neq f(N(F),A)$.

<u>Definition 7.5</u> A <u>pure output cell fault</u> (POCF), F_p, for a network of MOS cells is a single or multiple fault which involves FET's only in the output cell.

<u>Definition 7.6</u> A <u>non-pure output cell fault</u> (NPOCF), F_n, for a network of MOS cells is a single or multiple fault which involves at least one FET which is not in the output cell.

Let Φ_p denote the set of all possible POCF's and Φ_n denote the set of all possible NPOCF's in a network N. Obviously, the set of all possible single and multiple faults, Φ_t, is the union of Φ_p and Φ_n.

<u>Definition 7.7</u> A <u>sufficient test set</u>, Δ, for a class of faults, Φ, in a network N of MOS cells is a set of input vectors, $\Delta \subseteq V_n$, such that for each fault $F \in \Phi$ there exists an input vector, $A \in \Delta$, satisfying $f(N,A) \neq f(N(F),A)$, i.e., A is a test for N(F).

<u>Definition 7.8</u> An <u>inverse pair</u> in $C_n(\tilde{f})$ is an ordered pair of vertices, (A,B), $A,B \in C_n$ such that:

(1) $A > B$;

(2) $\tilde{f}(A) = 1$ and $\tilde{f}(B) = 0$; and

(3) For each vertex C for which $A > C > B$, $\tilde{f}(C) = *$.

Note that in the special case where $\overrightarrow{A\,B}$ is an inverse edge, (A,B) is an inverse pair.

Definition 7.9 The characteristic input set of an incompletely specified function $\tilde{f}$, denoted $S_c(\tilde{f})$, is the set of vectors each of which is in at least one inverse pair of $C_n(\tilde{f})$.

The following theorem is demonstrated in [Lai 76]. Algorithm DIMN (from [Lai 76]) will be given later.

Theorem 7.1 The characteristic input set, $S_c(\tilde{f})$, for an incompletely specified function $\tilde{f}$ is a sufficient test set for the set of all possible non-pure output cell faults, Φ_n, for a network N synthesized by Algorithm DIMN for $\tilde{f}$.

The next theorem, also appearing in [Lai 76], is clear from preceding definitions.

Theorem 7.2 Set $S = \{A|\tilde{f}(A) \neq *\}$, i.e., the set of specified vectors for $\tilde{f}$, is a sufficient test set for all possible single or multiple faults, Φ_t, in a network N synthesized by Algorithm DIMN for $\tilde{f}$.

Although this theorem gives the entire set of 2^n input vectors, V_n, as a sufficient test set when $\tilde{f}$ is a completely specified function, non-irredundant networks may require more than 2^n tests (auxiliary test points are needed). For example, one network discussed in [Pai 73] for which n = 3 requires nine tests for the output cell alone although the number of different input vectors is only $2^3 = 8$.

Algorithm DIMN can, in general, synthesize more than one irredundant network to realize a given function $\tilde{f}$. Test sets given in Theorems 7.1 and 7.2, however, depend only on $\tilde{f}$ and not on the specific synthesized network. In this sense, these test sets are 'universal.'

Example 7.1 A completely specified function f is shown in Fig. 7.1.[†] By Theorem 7.1, a sufficient test set for Φ_n in any network N synthesized by Algorithm DIMN for f consists of 13 input vectors: {(1111), (1101), (1011), (0111), (1100), (1010), (0110), (1001), (1000), (0100), (0010), (0001), (0000)}. By Theorem 7.2, a sufficient test set for Φ_t in the same networks consists of all 16 input vectors in V_4. A considerably smaller sufficient test set for Φ_t in a particular network synthesized by Algorithm DIMN for f will be derived in a later example through the use of a proposed test set generation algorithm.

Suppose there exist two functions, $\tilde{f}_1$ and $\tilde{f}_2$, related in the following manner: (1) for all vectors $A \in V_n$ which are specified vectors of both $\tilde{f}_1$ and $\tilde{f}_2$, $\tilde{f}_1(A) = \tilde{f}_2(A)$; and (2) the number of specified vectors of $\tilde{f}_1$ exceeds the number of specified vectors of $\tilde{f}_2$. Further suppose that Algorithm DIMN can synthesize the identical irredundant network N for both $\tilde{f}_1$ and $\tilde{f}_2$. Then, by Theorem 7.2, the set of specified vectors for $\tilde{f}_2$ is a sufficient test set for Φ_t in N. Hence, although $\tilde{f}_1$ may have many more specified vectors than $\tilde{f}_2$ and the sufficient test set given by Theorem 7.2 for network N synthesized for $\tilde{f}_1$ would include all of these specified vectors, the smaller set of specified vectors for $\tilde{f}_2$ will actually suffice. Therefore, the problem of finding a smaller sufficient test set for Φ_t (than that given by Theorem 7.2) in a network N synthesized by Algorithm DIMN for a function $\tilde{f}_1$ can be solved by finding a function $\tilde{f}_2$, having fewer specified vectors, for which Algorithm DIMN can

[†] The number shown above each vertex is the input vector corresponding to that vertex.

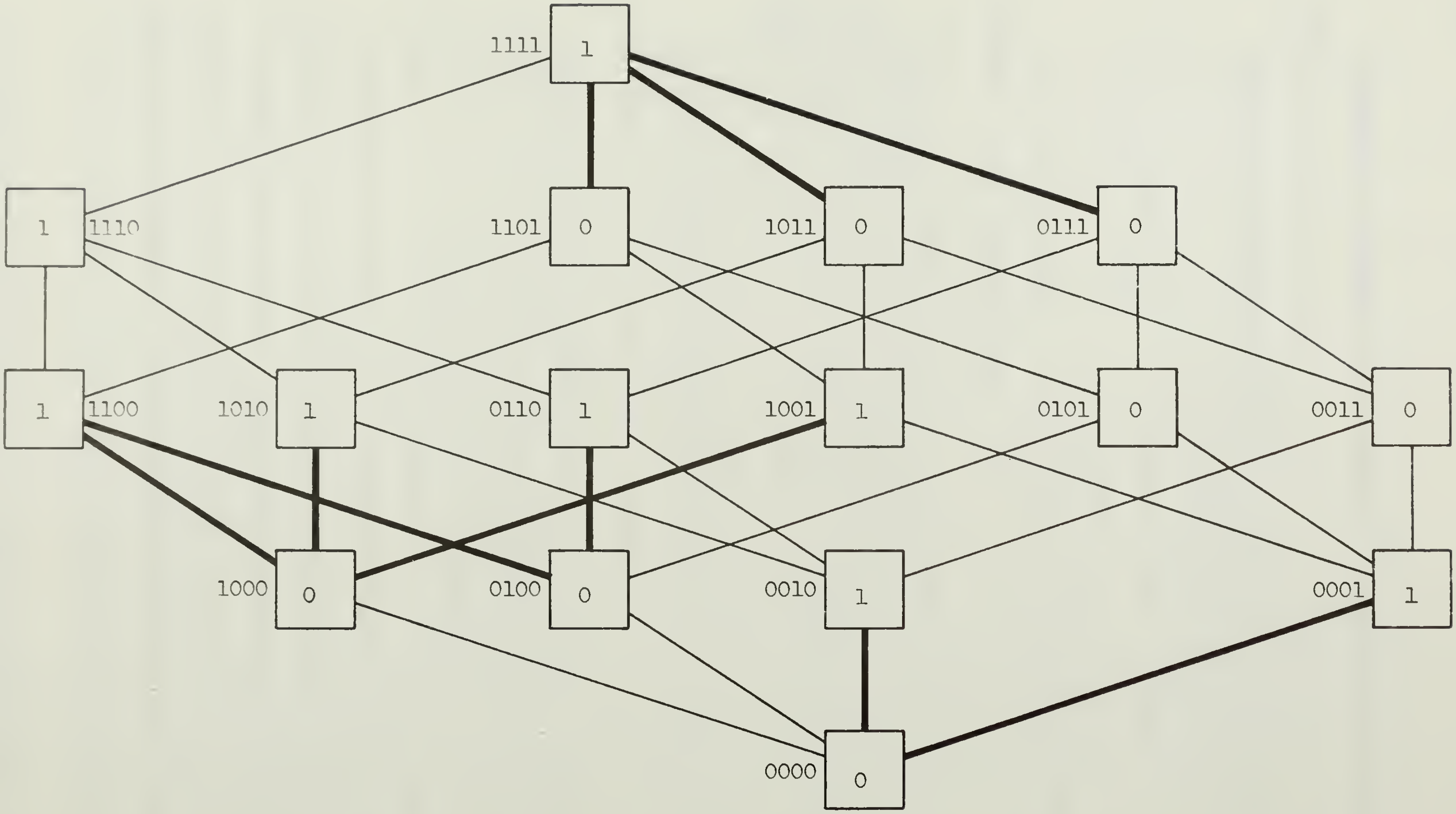


Fig. 7.1 Function f for Example 7.1. (Inverse edges shown in bold lines.)

synthesize the same network N. (Note that if A is a specified vector for both $\tilde{f}_1$ and $\tilde{f}_2$, then $\tilde{f}_1(A) = \tilde{f}_2(A) = f(N,A)$ must hold.)

This approach was suggested in [Lai 76] although the problem of finding such an $\tilde{f}_2$ for a given $\tilde{f}_1$ was left unsolved. An algorithm for generating reduced test sets based on this principle will be presented here after the introduction of Algorithm DIMN. First, definitions and notation necessary to Algorithm DIMN (and appearing in [Lai 76]) are given:

Definition 7.10 A partially specified negative function sequence of length R[†] and degree i for an incompletely specified function $\tilde{f}$ of n variables is a sequence of R functions, denoted by $NFS_n^i(R,\tilde{f}) = (u_1,\ldots,u_i, u_{i+1}^*,\ldots, u_{R-1}^*, \tilde{u}_R = \tilde{f})$, such that:

(1) $u_1,\ldots,u_i$ are completely specified functions with respect to $x_1,\ldots,x_n$;

(2) $(u_1,\ldots,u_i)$ is a $NFS_n(i,u_i)$;

(3) $u_{i+1}^*,\ldots,u_{R-1}^*$ are completely unspecified functions (i.e., all components of the vectors representing these functions are *'s); and

(4) $\tilde{u}_R = \tilde{f}$ is an incompletely specified function with respect to $x_1,\ldots,x_n$.

A set of complete specifications, $u_{i+1},\ldots,u_R = f$, of $u_{i+1}^*,\ldots,u_{R-1}^*$, $\tilde{u}_R = \tilde{f}$, with respect to $x_1,\ldots,x_n$, results in a completion of $NFS_n^i(R,\tilde{f})$, $(u_1,\ldots,u_R = f)$. A feasible $NFS_n^i(R,\tilde{f})$ is one for which there exists at least one completion of $NFS_n^i(R,\tilde{f})$ which is a $NFS_n(R,f)$. Such a completion is called a feasible completion of $NFS_n^i(R,\tilde{f})$. Any completion which is not an $NFS_n(R,f)$ is called an infeasible completion of $NFS_n^i(R,\tilde{f})$, and an infeasible $NFS_n^i(R,\tilde{f})$ is one for which every completion is an infeasible completion.

[†] Only the case in which $R = R_{\tilde{f}}$ will be important for subsequent discussion.

The following three algorithms, given in [Lai 76], will be incorporated into Algorithm DIMN.

Algorithm 7.1 Algorithm for conditional minimum labelling of an n-cube for a feasible $NFS_n^i(R,\tilde{f})$ (CMNL).

The conditional minimum labelling of C_n for a feasible $NFS_n^i(R,\tilde{f})$ ($R = R_{\tilde{f}}$) is a feasible completion of $NFS_n^i(R,\tilde{f})$, denoted $\underline{NFS}_n^i(R,\tilde{f}) = (u_1,\ldots,u_i, \underline{u}_{i+1}, \ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$, for which, in $C_n(u_1,\ldots,u_i,\underline{u}_{i+1},\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$, the label $\ell(A; u_1,\ldots,u_i, \underline{u}_{i+1},\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$, denoted $L_{mn}^{\tilde{f},i}(A)$, assumes for every $A \in C_n$ the minimum possible value among all feasible completions of $NFS_n^i(R,\tilde{f})$.

Step 1 If $\tilde{f}(\vec{1}) = *$, assign $L_{mn}^{\tilde{f},i}(\vec{1}) = \sum_{k=1}^{i} 2^{R-k}u_k(\vec{1})$; otherwise, assign $L_{mn}^{\tilde{f},i}(\vec{1}) = \sum_{k=1}^{i} 2^{R-k}u_k(\vec{1}) + \tilde{f}(\vec{1})$. Set $w = n$.

Step 2 $w = w - 1$. If $w < 0$, go to Step 4.

Step 3 For each vertex A of weight w, let Q^A be the set of all vectors of weight $w + 1$ such that $B > A$ for every $B \in Q^A$, and then assign as $L_{mn}^{\tilde{f},i}(A)$ the smallest binary integer satisfying the three conditions:

(i) The k-th most significant bit of $L_{mn}^{\tilde{f},i}(A)$ is $u_k(A)$, $k = 1,\ldots,i$;

(ii) If $\tilde{f}(A) \neq *$, the least significant bit of $L_{mn}^{\tilde{f},i}(A)$ is $\tilde{f}(A)$; and

(iii) $L_{mn}^{\tilde{f},i}(A) \geq L_{mn}^{\tilde{f},i}(B)$ for every $B \in Q^A$.

Go to Step 2.

Step 4 The k-th most significant bit and the least significant bit of $L_{mn}^{\tilde{f},i}(A)$ are denoted $\underline{u}_k(A)$ and $\underline{\tilde{f}}(A)$, respectively, for each $A \in C_n$. The feasible completion $\underline{NFS}_n^i(R,\tilde{f}) = (u_1,\ldots,u_i, \underline{u}_{i+1},\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$ of $NFS_n^i(R,\tilde{f})$ has been obtained.

Algorithm 7.2 Algorithm for conditional maximum labelling of an n-cube for a feasible $NFS_n^i(R,\tilde{f})$ (CMXL).

The conditional maximum labelling of C_n for a feasible $NFS_n^i(R,\tilde{f})$ ($R = R_{\tilde{f}}$) is a feasible completion of $NFS_n^i(R,\tilde{f})$, denoted $\widehat{NFS}_n^i(R,\tilde{f}) = (u_1,\ldots,u_i,\hat{u}_{i+1},\ldots,\hat{u}_{R-1},\hat{\tilde{f}})$, for which, in $C_n(u_1,\ldots,u_i,\hat{u}_{i+1},\ldots,\hat{u}_{R-1},\hat{\tilde{f}})$, the label $\ell(A;u_1,\ldots,u_i,\hat{u}_{i+1},\ldots,\hat{u}_{R-1},\hat{\tilde{f}})$, denoted, $L_{mx}^{\tilde{f},i}(A)$, assumes for every $A \in C_n$ the maximum possible value among all feasible completions of $NFS_n^i(R,\tilde{f})$.

Step 1 If $\tilde{f}(\vec{0}) = *$, assign $L_{mx}^{\tilde{f},i}(\vec{0}) = \sum_{k=1}^{i} 2^{R-k}u_k(\vec{0}) + 2^{R-i} - 1$; otherwise assign $L_{mx}^{\tilde{f},i}(\vec{0}) = \sum_{k=1}^{i} 2^{R-k}u_k(\vec{0}) + 2^{R-i} - 2 + \tilde{f}(\vec{0})$. Set $w = 0$.

Step 2 $w = w + 1$. If $w > n$, go to Step 4.

Step 3 For each vertex A of weight w, let Q_A be the set of all vectors of weight $w - 1$ such that $A > B$ for every $B \in Q_A$, and then assign as $L_{mx}^{\tilde{f},i}(A)$ the largest binary integer satisfying the three conditions:

(i) The k-th most significant bit of $L_{mx}^{\tilde{f},i}(A)$ is $u_k(A)$, $k = 1,\ldots,i$;

(ii) If $\tilde{f}(A) \neq *$, the least significant bit of $L_{mx}^{\tilde{f},i}(A)$ is $\tilde{f}(A)$; and

(iii) $L_{mx}^{\tilde{f},i}(A) \leq L_{mx}^{\tilde{f},i}(B)$ for every $B \in Q_A$.

Go to Step 2.

Step 4 The k-th most significant bit and the least significant bit of $L_{mx}^{\tilde{f},i}(A)$ are denoted $\hat{u}_k(A)$ and $\hat{\tilde{f}}(A)$, respectively, for each $A \in C_n$. The feasible completion $\widehat{NSF}_n^i(R,\tilde{f}) = (u_1,\ldots,u_i, \hat{u}_{i+1},\ldots,\hat{u}_{R-1}, \hat{\tilde{f}})$ of $NFS_n^i(R,\tilde{f})$ has been obtained.

Definition 7.11 The maximum permissible function $\tilde{u}_{i+1}$ for a feasible partially specified negative function sequence of length $R_{\tilde{f}}$ and degree i for function $\tilde{f}$, $NFS_n^i(R_{\tilde{f}},\tilde{f}) = (u_1,\ldots,u_i, \overset{*}{u}_{i+1},\ldots,\overset{*}{u}_{R_{\tilde{f}}-1},\tilde{f})$ is an incompletely specified function for $\overset{*}{u}_{i+1}$ such that:

(1) $\tilde{u}_{i+1}$ is negative with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_i$;

(2) every negative completion u_{i+1} of $\tilde{u}_{i+1}$ yields a feasible $NFS_n^{i+1}(R_f,f)$ $= (u_1,\ldots,u_{i+1}, u^*_{i+2},\ldots,u^*_{R_f-1}, \tilde{f})$; and

(3) any function u'_{i+1} which is not a negative completion of $\tilde{u}_{i+1}$ yields an infeasible $NFS_n^{i+1}(R_f,f) = (u_1,\ldots,u_i, u'_{i+1}, u^*_{i+2},\ldots,u^*_{R_f-1}, \tilde{f})$.

Algorithm 7.3 Algorithm to obtain the maximum permissible function $\tilde{u}_{i+1}$ for a given feasible $NFS_n^i(R,\tilde{f}) = (u_1,\ldots,u_i, u^*_{i+1},\ldots,u^*_{R-1}, \tilde{f})$ (MPF).

Step 1 Obtain the completion $\underline{NFS}_n^i(R,\tilde{f}) = (u_1,\ldots,u_i, \underline{u}_{i+1},\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$ of $NFS_n^i(R,\tilde{f})$ according to Algorithm CMNL.

Step 2 Obtain the completion $\widehat{NFS}_n^i(R,\tilde{f}) = (u_1,\ldots,u_i, \hat{u}_{i+1},\ldots,\hat{u}_{R-1}, \hat{\tilde{f}})$ of $NFS_n^i(R,\tilde{f})$ according to Algorithm CMXL.

Step 3 For each vertex $A \in C_n$:

(a) assign $\tilde{u}_{i+1}(A)$ the value 0 if and only if $\underline{u}_{i+1}(A) = \hat{u}_{i+1}(A) = 0$;

(b) assign $\tilde{u}_{i+1}(A)$ the value 1 if and only if $\underline{u}_{i+1}(A) = \hat{u}_{i+1}(A) = 1$;

(c) otherwise, assign $\tilde{u}_{i+1}(A)$ the value * (i.e., A is an unspecified vector for $\tilde{u}_{i+1}$).

The validity of these algorithms is demonstrated in [Lai 76].

On the basis of the preceding definitions and theorems, Algorithm DIMN of [Lai 76] can now be given:

Algorithm 7.4 Algorithm to design an irredendant MOS network with a minimum number of cells (G-minimal) for a given function $\tilde{f}$ (DIMN).

Step 1 Let $NFS_n^0(R,\tilde{f}) = (u^*_1,\ldots,u^*_{R-1}, \tilde{f})$. ($R = R_{\tilde{f}}$ is the minimum number of cells necessary to realize $\tilde{f}$. This value, if unknown, can be found during the first execution of Step 2, i.e., after applying Algorithm CMNL to obtain $\underline{NFS}_n^0(R,\tilde{f})$.) Set i = 0.

Step 2 Apply Algorithm MPF to obtain the maximum permissible function $\tilde{u}_{i+1}$ for $NFS_n^i(R,\tilde{f})$.

Step 3 Obtain a complemented irredundant disjunctive form or a complemented conjunctive form for a negative completion of $\tilde{u}_{i+1}$ with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_i$ such that the deletion of any literal, term, or alterm from the expression would result in a function which is not a negative completion of $\tilde{u}_{i+1}$ ([Iba 71] and [Lai 76] offer algorithms which produce such expressions). From this expression, create an MOS cell configured in the usual way (i.e., conjunctions (or terms) correspond to connections in series, disjunctions (or alterms) correspond to connections in parallel). Let u_{i+1} denote the function realized by this cell (completely specified with respect to $x_1,\ldots,x_n$).

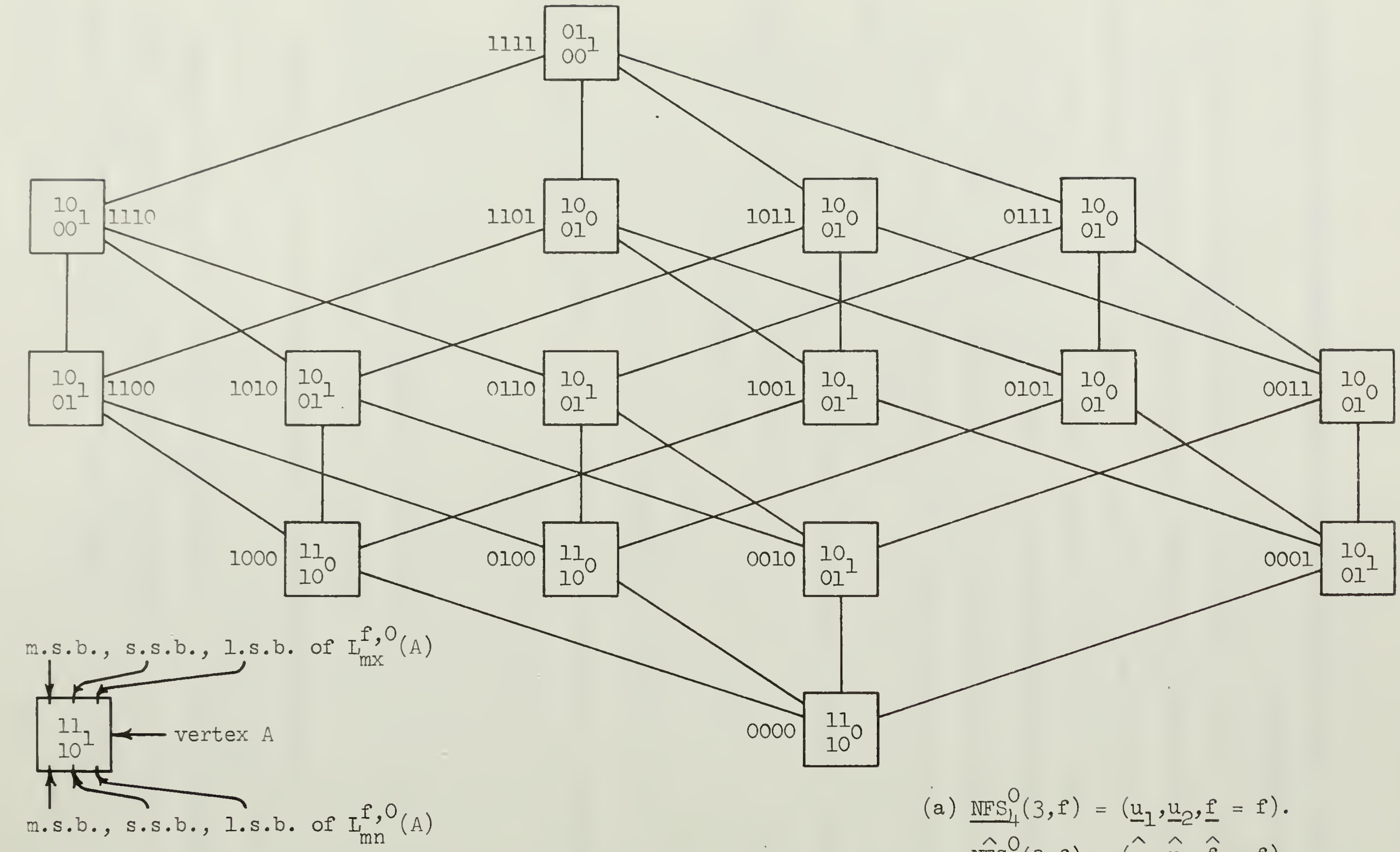
Step 4 If $i = R - 2$, create an MOS cell for $\tilde{f}$ as in Step 3 and terminate the algorithm. Otherwise, set $i = i + 1$ and return to Step 2.

The following theorem is demonstrated in [Lai 76].

Theorem 7.3 A network of MOS cells synthesized by Algorithm DIMN for a function $\tilde{f}$ is irredundant with respect to $\tilde{f}$.

Due to the flexibility in Step 3 of Algorithm DIMN, more than one irredundant network for a given $\tilde{f}$ can be synthesized in general. It should also be noted that the special expressions required in Step 3 are relatively easy to derive due to the fact that the functions involved are negative functions rather than general functions.

Example 7.2 Suppose an irredundant, G-minimal network of MOS cells is desired for function f of Example 7.1 (shown in Fig. 7.1). In Step 2 (which is actually Algorithm MPF) of Algorithm DIMN, the $\underline{NFS}_4^0(3,f)$ and $\widehat{NFS}_4^0(3,f)$ are

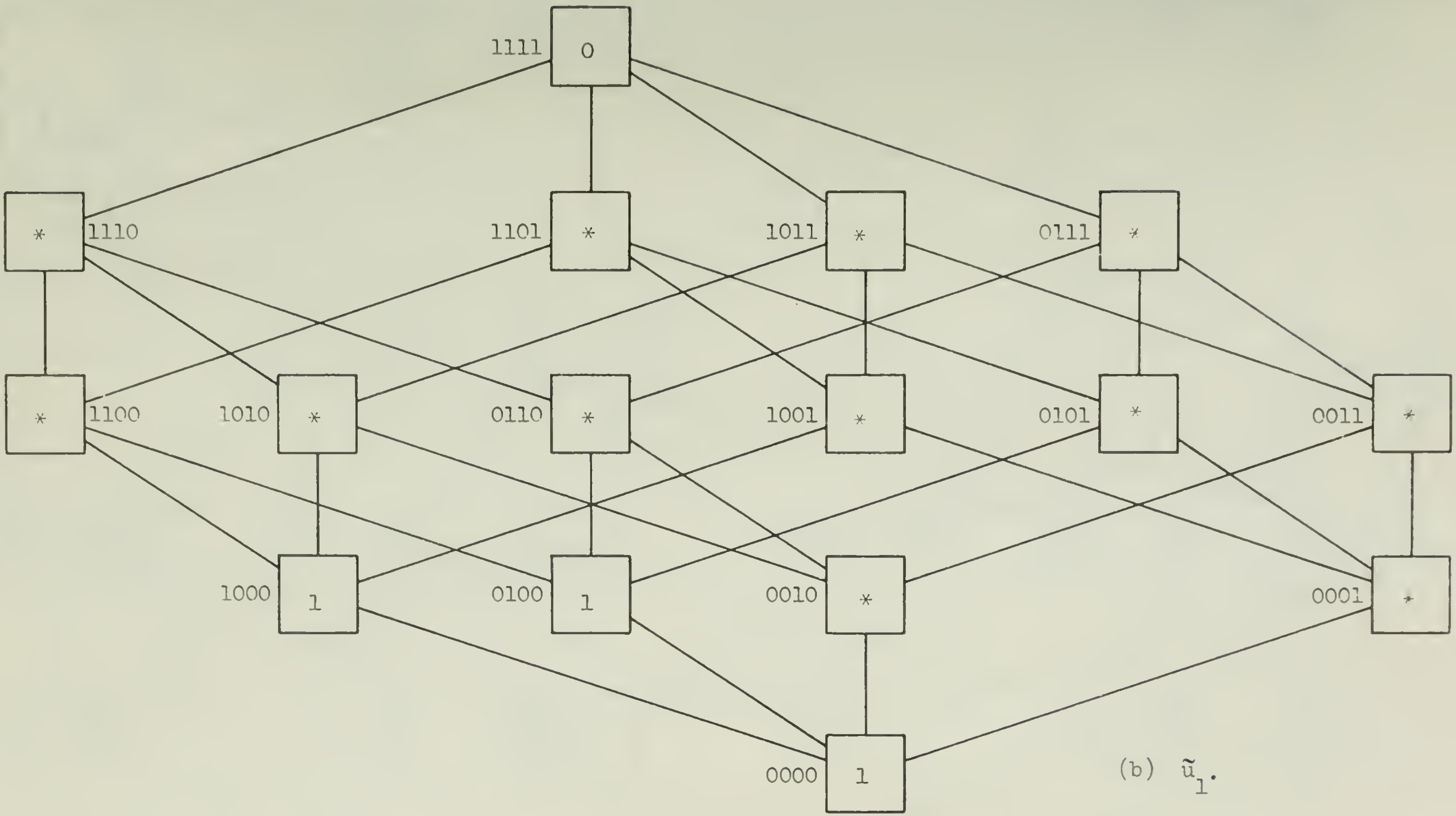


m.s.b. = most significant bit
s.s.b. = second (most) significant bit
l.s.b. = least significant bit

(a) $\underline{\mathrm{NFS}}_4^0(3,f) = (\underline{u}_1, \underline{u}_2, \underline{f} = f)$.

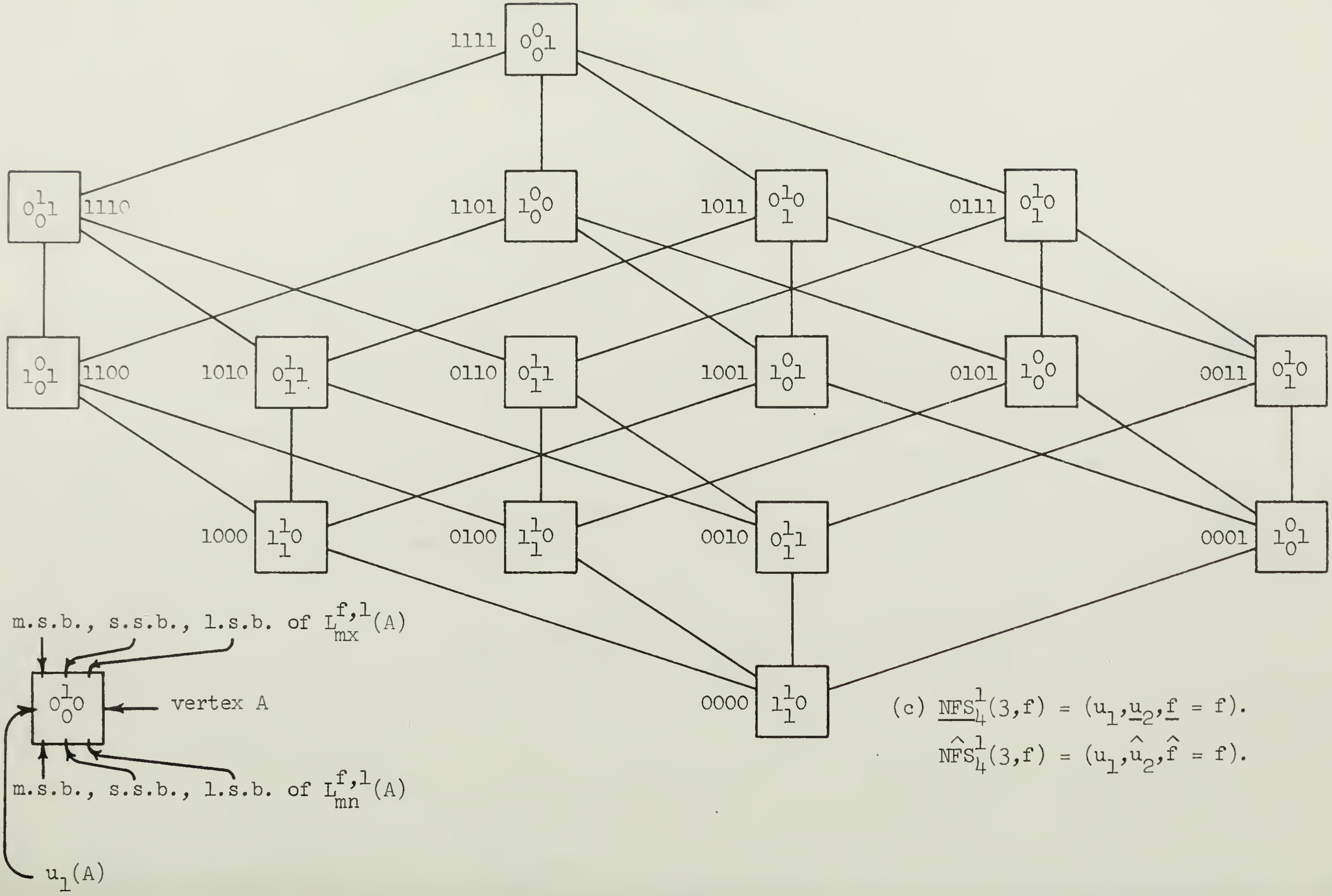
$\widehat{\mathrm{NFS}}_4^0(3,f) = (\hat{u}_1, \hat{u}_2, \hat{f} = f)$.

Fig. 7.2 Example 7.2.



(b) $\tilde{u}_1$.

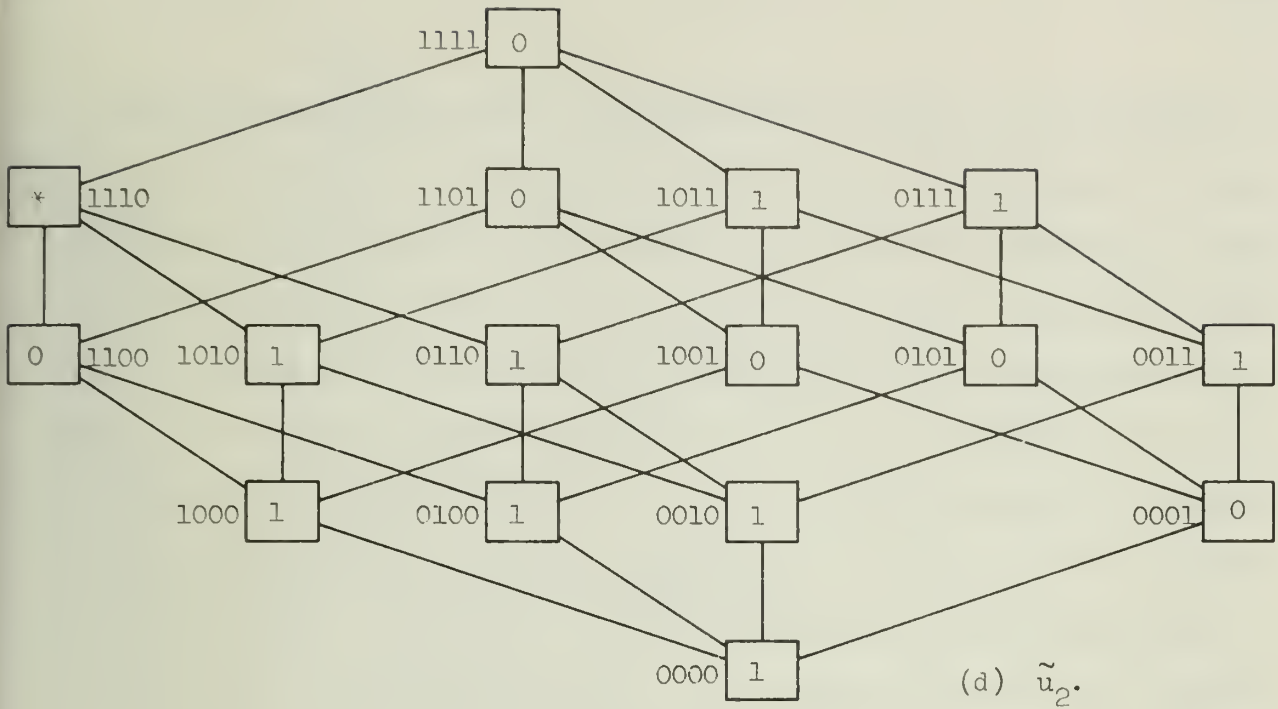
Fig. 7.2 (Continued)


1111
1110
1101
1011
0111
1100
1010
0110
1001
0101
0011
1000
0100
0010
0001
0000
m.s.b., s.s.b., l.s.b. of $L_{mx}^{f,1}(A)$
vertex A
m.s.b., s.s.b., l.s.b. of $L_{mn}^{f,1}(A)$
$u_1(A)$

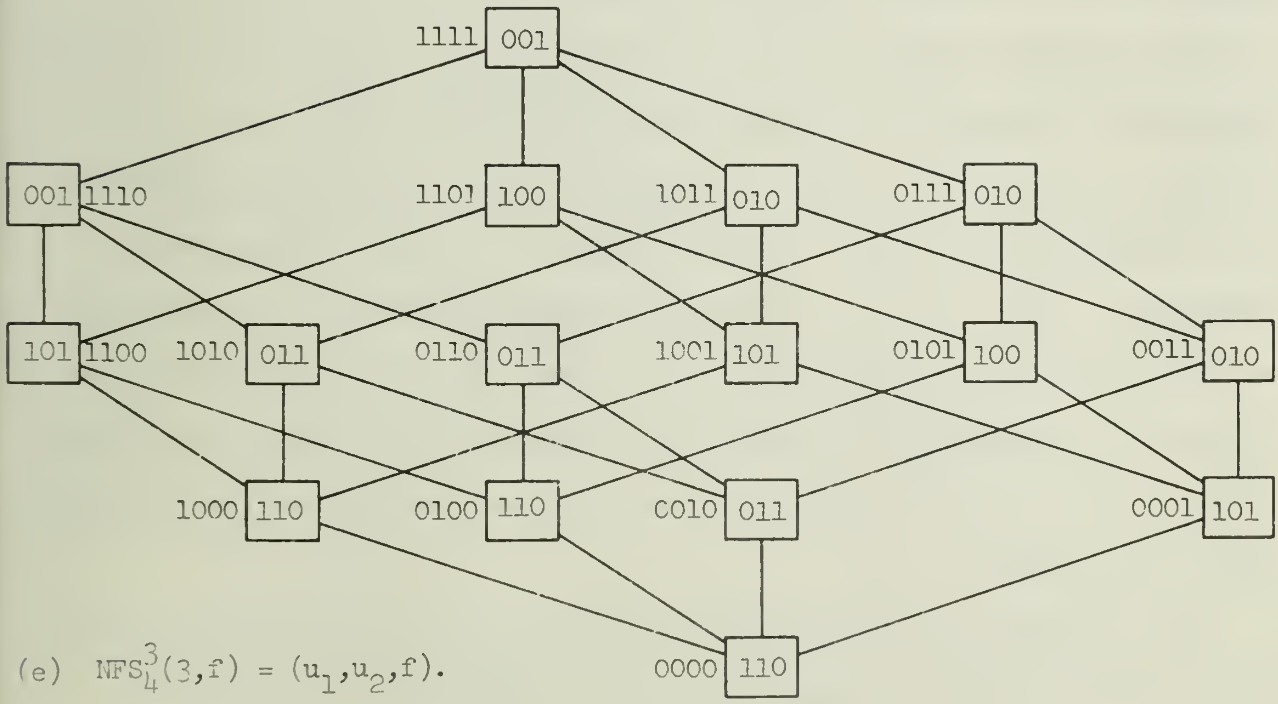
(c) $\underline{NFS}_4^1(3,f) = (u_1, \underline{u}_2, \underline{f} = f).$

$\widehat{NFS}_4^1(3,f) = (u_1, \hat{u}_2, \hat{f} = f).$

Fig. 7.2 (Continued)



(d) $\tilde{u}_2$.



(e) $NFS_4^3(3,f) = (u_1,u_2,f)$.

Fig. 7.2 (Continued)

first obtained, by Algorithms CMNL and CMXL, respectively, as shown in Fig. 7.2(a). A comparison of $\underline{u}_1$ and $\hat{u}_1$ then results in $\tilde{u}_1$ as shown in (b). In Step 3 of Algorithm DIMN, any of the three expressions $(\bar{x}_3)$, $(\bar{x}_4)$, or $(\overline{x_1x_2})$ could be selected. Assume $\bar{x}_3$ is chosen: $u_1 = \bar{x}_3$. The corresponding MOS cell configuration will be given later. Returning to Step 2, $\underline{NFS}_4^1(3,f) = (\bar{x}_3, \underline{u}_2, \underline{f} = f)$ and $\widehat{NFS}_4^1(3,f) = (\bar{x}_3, \hat{u}_2, \hat{f} = f)$ are obtained as shown in (c). The resulting $\tilde{u}_2$ is given in (d). In Step 3, let $\overline{x_1x_2 \vee x_4u_1}$ be the expression chosen for the second cell of the network. Since $R_f = 3$, $NFS_4^2(3,f) = (u_1, u_2, f)$ is a completely specified minimum NFS with respect to x_1, x_2, x_3 (see (e)). In Step 4, the synthesis is completed, letting $\overline{x_2x_4u_1 \vee x_4u_2 \vee u_1u_2}$ be the expression selected for the output cell. The corresponding G-minimal, irredundant network of MOS cells is shown in Fig. 7.3. The sufficient test sets for Φ_n and Φ_t determined in Example 7.1 can be used to diagnose this network (as well as other possible networks resulting from Algorithm DIMN for f).

The sufficient test set (for Φ_t) generation method which will be proposed can be applied subsequent to the synthesis of a network N by Algorithm DIMN (the test set generation algorithm to be given will be stated for application in this manner), but it will be most efficient if combined with Algorithm DIMN itself. This is due to the fact that certain information derived during the application of Algorithm DIMN is also needed for the algorithm generating the test set. A generated sufficient test set will not, in general, be a sufficient test set for networks other than a particular N generated by Algorithm DIMN for the given function.

The following definitions will be needed for the presentation of the proposed test set generation algorithm, Algorithm TSG.

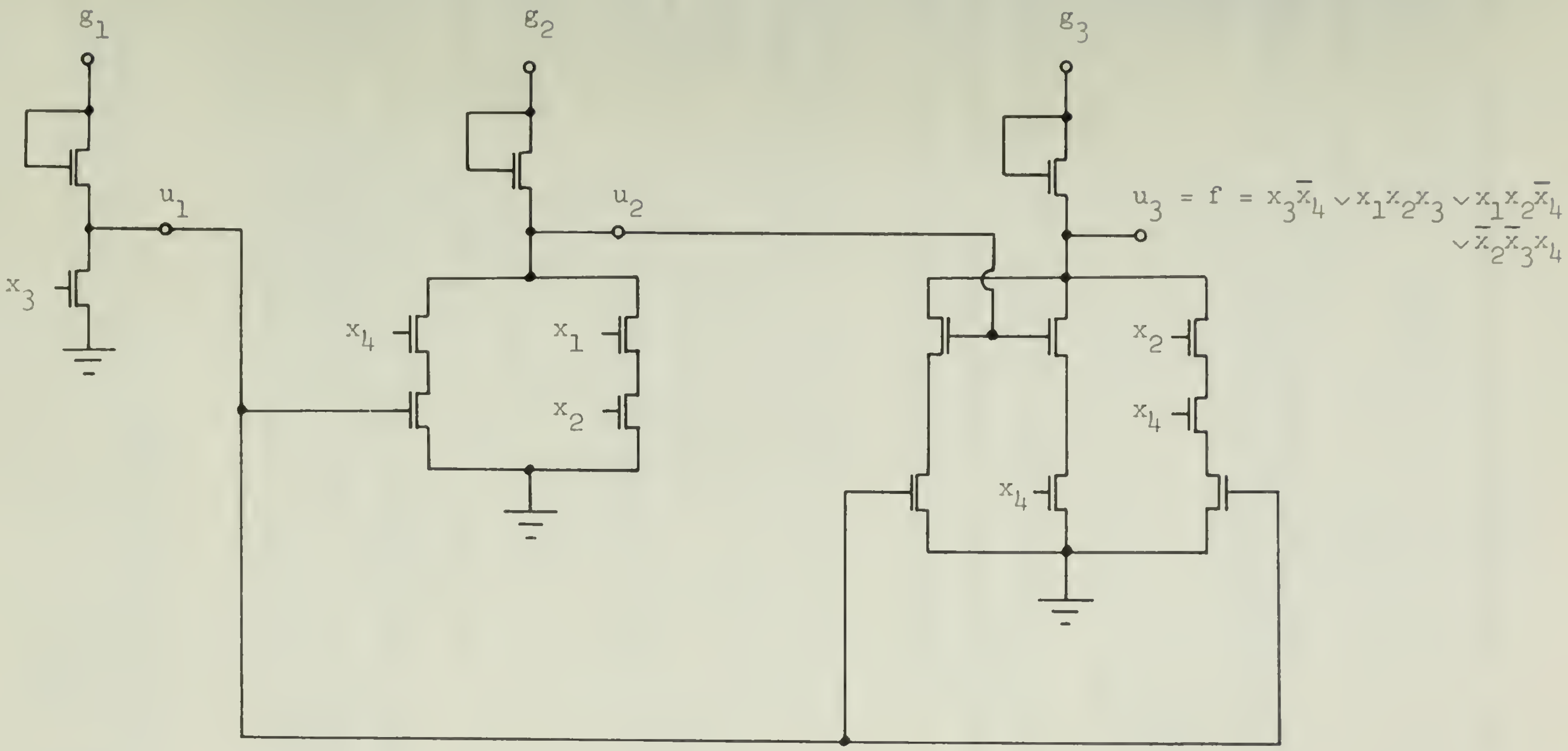

g_1
g_2
g_3
u_1
u_2
$u_3 = f = x_3\bar{x}_4 \vee x_1x_2x_3 \vee x_1x_2\bar{x}_4 \vee \bar{x}_2\bar{x}_3x_4$
x_3
x_4
x_1
x_2
x_2
x_4
x_4

Fig. 7.3 G-minimal, irredundant network synthesized by Algorithm DIMN in Example 7.2.

Definition 7.12 For a given $\underline{\mathrm{NFS}}_n^i(R,\tilde{f}) = (u_1,\ldots,u_i, \underline{u}_{i+1},\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$ and corresponding labelled n-cube, $C_n(u_1,\ldots,u_i, \underline{u}_{i+1},\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$, in which every vertex $A \in C_n$ has the label $L_{mn}^{\tilde{f},i}(A) = \ell(A; u_1,\ldots,u_i, \underline{u}_{i+1},\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$, a vertex B is said to be an upper-supportive vertex (USV) of a vertex A, with respect to the conditional minimum labelling, if and only if:

(i) $\overrightarrow{BA}$ is an edge of C_n;

(ii) $L_{mn}^{\tilde{f},i}(A) > \sum_{k=1}^{i} 2^{R-k} u_k(A) + 1$; and

(iii) The label, L'(A), which would result from Step 3 of Algorithm CMNL for vertex A if the label $L_{mn}^{\tilde{f},i}(C)$ were 0 for every vertex C, $C \neq B$, in the set $\{C | \overrightarrow{CA}$ is an edge of $C_n\}$ ($L_{mn}^{\tilde{f},i}(B)$ retaining its original value), satisfies $L'(A) = L_{mn}^{\tilde{f},i}(A)$.

Not every vertex in such a labelled n-cube has a USV, and some may have more than one. Obviously, the vertex $\vec{1}$ has no USV since no vertex B satisfying $\overrightarrow{B1}$ exists. In Example 7.3 which will be given after the following definition, examples of other vertices without USV's can be seen.

Definition 7.13 For a given $\underline{\mathrm{NFS}}_n^i(R,\tilde{f}) = (u_1,\ldots,u_i, \underline{u}_{i+1},\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$ and corresponding labelled n-cube, $C_n(u_1,\ldots,u_i, \underline{u}_{i+1},\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$, in which every vertex $A \in C_n$ has the label $L_{mn}^{\tilde{f},i}(A) = \ell(A; u_1,\ldots,u_i, \underline{u}_{i+1},\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$, a sequence of vertices $(B_1,\ldots,B_\ell)$ is said to be an upper-supportive chain (USC) of a vertex A, with respect to the conditional minimum labelling, if and only if:

(i) $\overrightarrow{B_1B_2}, \overrightarrow{B_2B_3},\ldots,\overrightarrow{B_{\ell-1}B_\ell}, \overrightarrow{B_\ell A}$ are edges of C_n;

(ii) B_i is an upper-supportive vertex of B_{i+1}, $i = 1,\ldots,\ell - 1$, and B_ℓ is an upper-supportive vertex of A; and

(iii) No upper-supportive vertex exists for B_1.

If a vertex has no USV, its USC is the null chain. Several USC's may exist for the same vertex.

Example 7.3 The conditional minimum labelling corresponding to $\underline{NFS}^0_4(3,f)$ in Example 7.2 (see Fig. 7.2.(a)) is reproduced in Fig. 7.4. All USV's for vertices in the 4-cube are shown by arrows (not to be confused with directed or inverse edges) from the USV to the vertex it "supports." Only vertices (1111) and (1110) have no USV's. Several vertices (e.g., (1000), (0100)) have three or more USV's. Two of several possible USC's for vertex (0100) are: ((1111), (1101), (1100)) and ((1111), (0111), (0110)).

Definition 7.14 For a given $\widehat{NFS}^i_n(R,\tilde{f}) = (u_1,\dots,u_i, \hat{u}_{i+1},\dots,\hat{u}_{R-1}, \hat{\tilde{f}})$ and corresponding labelled n-cube, $C_n(u_1,\dots,u_i, \hat{u}_{i+1},\dots,\hat{u}_{R-1}, \hat{\tilde{f}})$, in which every vertex $A \in C_n$ has the label $L^{\tilde{f},i}_{mx}(A) = \ell(A; u_1,\dots,u_i, \hat{u}_{i+1},\dots,\hat{u}_{R-1}, \hat{\tilde{f}})$, a vertex B is said to be a lower-supportive vertex (LSV) of a vertex A, with respect to the condtional maximum labelling, if and only if:

(i) $\overrightarrow{AB}$ is an edge of C_n;

(ii) $L^{\tilde{f},i}_{mx} < \sum_{k=1}^{i} 2^{R-k} u_k(A) + 2^{R-i} - 2$; and

(iii) The label, L'(A), which would result from Step 3 of Algorithm CMXL for vertex A if the label $L^{\tilde{f},i}_{mx}(C)$ were $2^R - 1$ for every vertex C, $C \neq B$, in the set $\{C \mid \overrightarrow{AC}$ is an edge of $C_n\}$ ($L^{\tilde{f},i}_{mx}(B)$ retaining its original value), satisfies $L'(A) = L^{\tilde{f},i}_{mx}(A)$.

Definition 7.15 For a given $\widehat{NFS}^i_n(R,\tilde{f}) = (u_1,\dots,u_i, \hat{u}_{i+1},\dots,\hat{u}_{R-1}, \hat{\tilde{f}})$ and corresponding labelled n-cube, $C_n(u_1,\dots,u_i, \hat{u}_{i+1},\dots,\hat{u}_{R-1}, \hat{\tilde{f}})$, in which every vertex $A \in C_n$ has the label $L^{\tilde{f},i}_{mx}(A) = \ell(A; u_1,\dots,u_i, \hat{u}_{i+1},\dots,\hat{u}_{R-1}, \hat{\tilde{f}})$, a sequence of vertices $(B_1,\ldots,B_\ell)$ is said to be a lower-supportive chain (LSC) of a vertex A, with respect to the conditional maximum labelling, if and only if:

(i) $\overrightarrow{AB_\ell}$, $\overrightarrow{B_\ell B_{\ell-1}}$, $\overrightarrow{B_{\ell-1}B_{\ell-2}},\ldots,\overrightarrow{B_2B_1}$ are edges of C_n;

(ii) B_i is a lower-supportive vertex of B_{i+1}, $i = 1,\ldots,\ell - 1$, and B_ℓ is a lower-supportive vertex of A; and

(iii) No lower-supportive vertex exists for B_1.

If a vertex has no LSV, its LSC is the null chain.

Example 7.4 The conditional maximum labelling corresponding to $\widehat{NFS}_4^0(3,f)$ in Example 7.2 (see Fig. 7.2(a)) is reproduced in Fig. 7.5. All LSV's for vertices in the 4-cube are shown by arrows from the LSV to the vertex it 'supports.' Only vertices (1000), (0100), and (0000) have no LSV's. Several vertices have three LSV's. One LSC for (1111) is: ((1000), (1010), (1011)).

The four preceding definitions are actually slightly more general than they need be for use in Algorithm TSG. Alternative definitions will be discussed later, but these are somewhat more complex and less convenient for hand calculation.

Definition 7.16 Let $\tilde{u}_{i+1}$ be an incompletely specified negative function and u_{i+1} be a negative completion of $\tilde{u}_{i+1}$ with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_i$. A vertex $A \in C_{n+i}$ corresponding to vector $A \in V_{n+i}$ is said to be a branch-determinative vertex (BDV) for $\tilde{u}_{i+1}$ with respect to u_{i+1} if and only if:

(i) A is a (specified) false vector of $\tilde{u}_{i+1}$, i.e., $\tilde{u}_{i+1}(A) = 0$; and

(ii) If $\{B_1,\ldots,B_k\}$ $B_j \in V_{n+i}$, $j = 1,\ldots,k$, is the set of minimum false

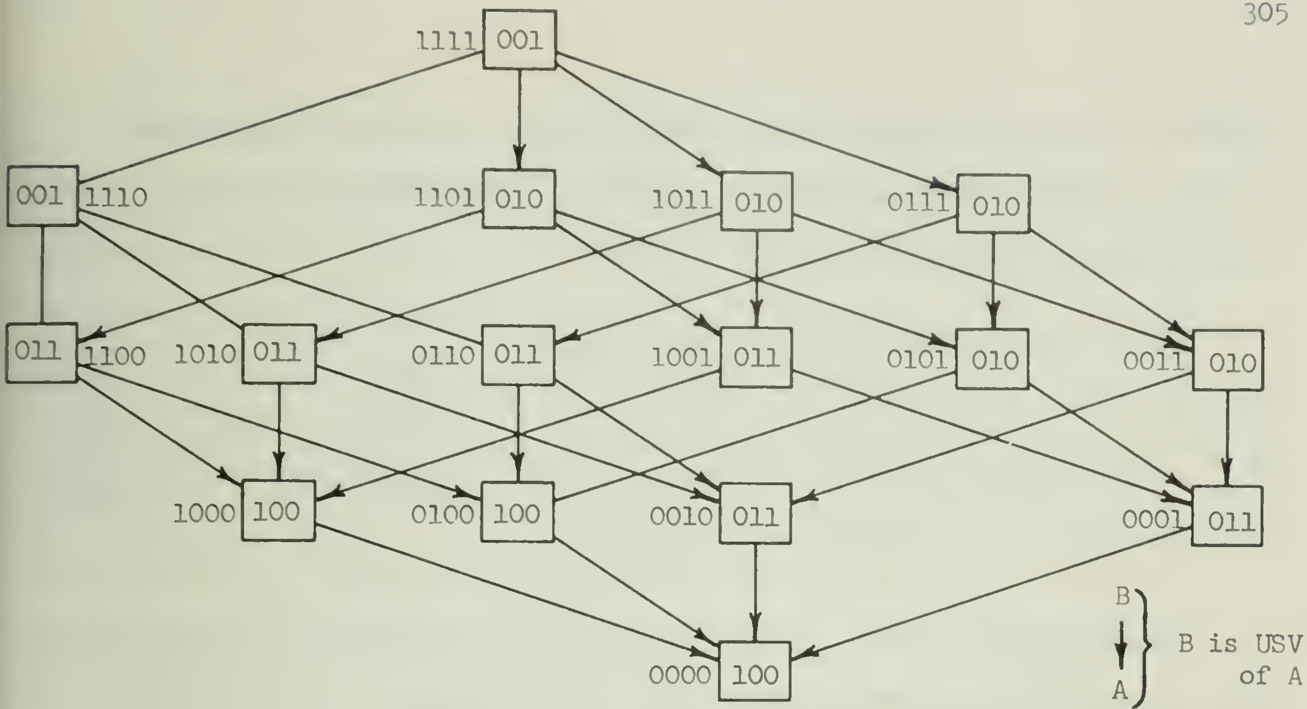


Fig. 7.4 Illustration of USV's for Example 7.3.

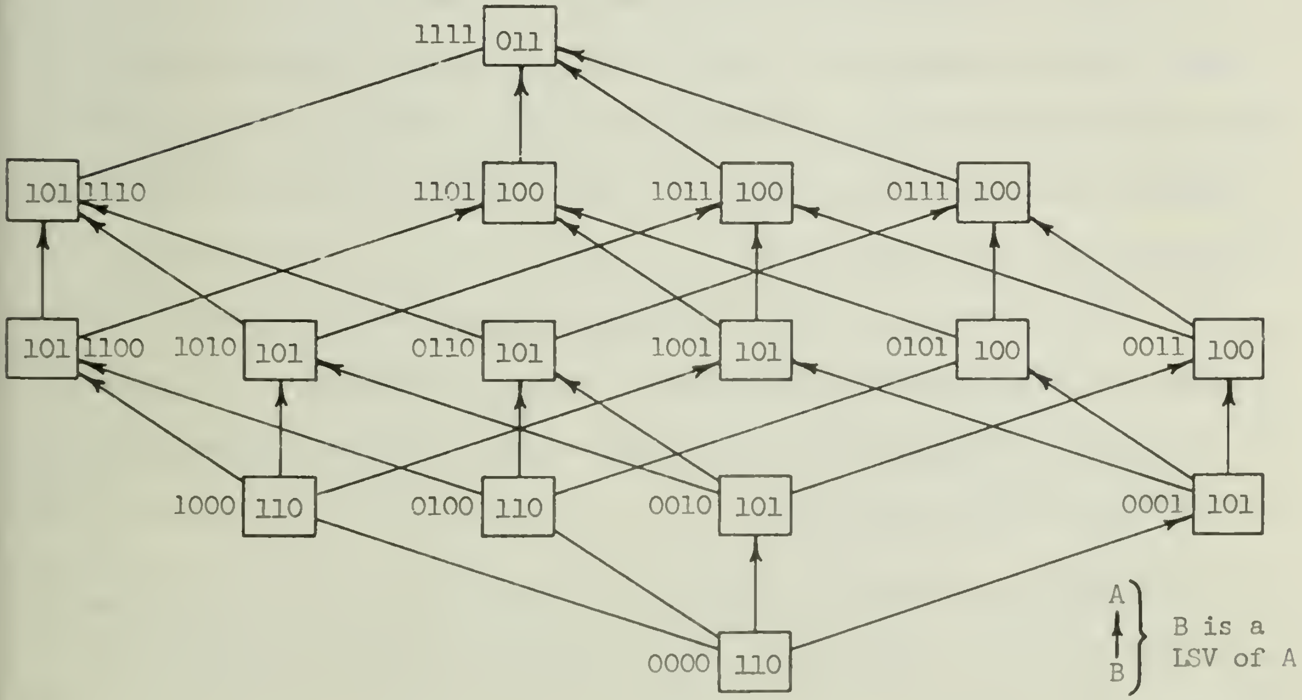


Fig. 7.5 Illustration of LSV's for Example 7.4.

vectors of u_{i+1}, $A \geq B_j$ for exactly one B_j, $1 \leq j \leq k$. (Vector B_j will be referred to as the minimum false vector of u_{i+1} corresponding to branch-determinative vector A.)

For an arbitrary negative completion, u_{i+1}, of $\tilde{u}_{i+1}$, BDV's may not exist. However, the concern here will be with certain 'irredundant' completions of $\tilde{u}_{i+1}$ (used in Step 3 of Algorithm DIMN) for which every minimum false vector will have one or more corresponding BDV's.

<u>Definition 7.17</u> Let $\tilde{u}_{i+1}$ be an incompletely specified negative function and u_{i+1} be a negative completion of $\tilde{u}_{i+1}$ with respect to $x_1,\ldots,x_n$, u_1, $\ldots,u_i$. A vertex $A \in C_{n+i}$ corresponding to vector $A \in V_{n+i}$ is said to be a <u>blocking vertex</u> (<u>BV</u>) of a branch-determinative vertex $B \in C_{n+i}$ for $\tilde{u}_{i+1}$ with respect to u_{i+1} if and only if:

(i) A is a (specified) true vector of $\tilde{u}_{i+1}$, i.e., $\tilde{u}_{i+1}(A) = 1$; and

(ii) For the minimum false vector $C \in V_{n+i}$ of u_{i+1} corresponding to B, at least one vector $D \in V_{n+i}$ exists such that $\overrightarrow{CD}$ is an edge in C_{n+i} and $D \leq A$.

A <u>complete set of blocking vertices</u> for branch-determinative vertex B is a set of vertices $\{A_1,\ldots,A_\ell\}$ such that:

(i) Each vertex A_j, $j = 1,\ldots,k$, is a blocking vertex of B; and

(ii) If C is the minimum false vector of u_{i+1} corresponding to B, then for every vector $D \in V_{n+i}$ such that $\overrightarrow{CD}$ is an edge in C_{n+i}, there exists at least one $A_j \in Q$ such that $D \leq A_j$.

A complete set of BV's for a BDV need be neither unique nor irredundant (in the sense that the removal of a vertex from the set would not result in a complete set of BV's).

Example 7.5 Consider the function $\tilde{u}_2$ (shown in Fig. 7.2(d) as a function of x_1, x_2, x_3, x_4) of Example 7.2 and its selected negative completion with respect to x_1, x_2, x_3, x_4, u_1, $u_2 = \overline{x_1x_2 \vee x_4u_1}$, of the same example. $\tilde{u}_2$ as an incompletely specified function of five variables, x_1, x_2, x_3, x_4, u_1, is shown in Fig. 7.6. Vectors (11110), (11011), (11001), (10011), (01011), and (00011) satisfy the first condition (see Definition 7.16) necessary for being a BDV for $\tilde{u}_2$. Since the minimum false vectors of u_2 are (11000) and (00011), vectors (11110), (11001), (10011), (01011), and (00011) also satisfy the second condition necessary for being a BDV for $\tilde{u}_2$. (11011) does not satisfy the second condition since there are two minimal false vectors less than vector (11011). Minimum false vector (11000) or u_2 corresponds BDV's (11110) and (11001). Minimum false vector (00011) of u_2 correspond BDV's (10011), (01011), and (00011). BV's of BDV (11110) are: (10110), (01110), (10100), (01100), (10001), and (01001). A complete set of BV's for BDV (11110) is {(10110),(01110)} (others also exist). BV's of BDV (10011) are: (10110), (01110), (10001), (00110), (01001), and (00001). One complete set of BV's for BDV (10011) is {(10110),(10001)}.

Algorithm TSG for the generation of a sufficient test set for all faults Φ_t of an irredundant network N synthesized by Algorithm DIMN can now be given. The algorithm is stated for the case in which the complement of an irredundant disjunctive form (rather than the complement of an irredundant conjunctive form) has been chosen for each $\tilde{U}_{i+1}$ in Step 3 of Algorithm DIMN during the synthesis of network N. The case in which the complemented irredundant conjunctive forms are chosen in Step 3 for some $\tilde{U}_{i+1}$ can be treated in a similar manner and will not be given explicitly here.

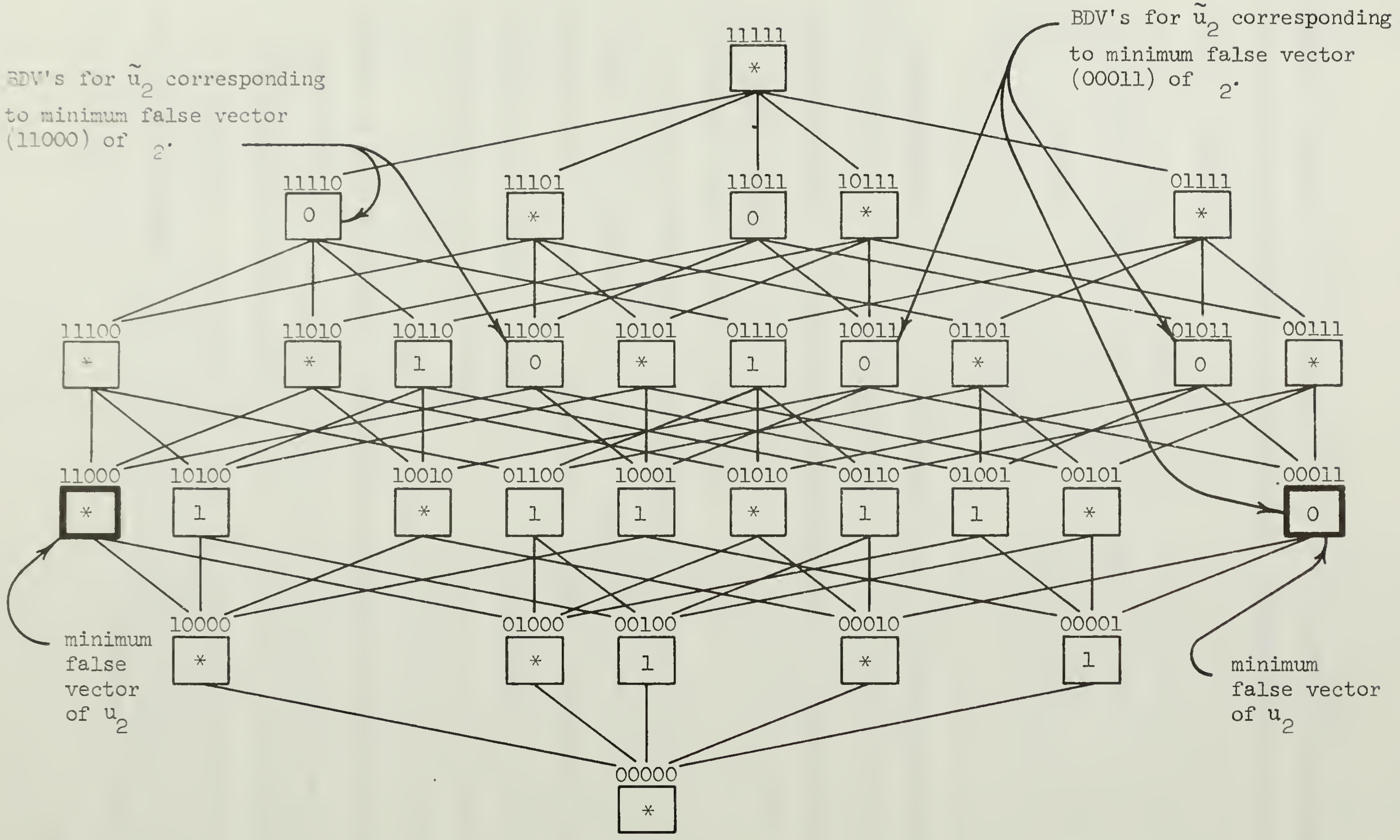


Fig. 7.6 Function $\tilde{U}_2$ of Example 7.5.

Algorithm 7.5 Algorithm to generate a sufficient test set for all possible single and multiple faults, Φ_t, in an irredundant network of MOS cells, N, synthesized by Algorithm DIMN for a given function $\tilde{f}$ (TSG).

Let $u_1, \ldots u_R$ be the functions for which the complemented irredundant disjunctive forms were selected in Step 3 of Algorithm DIMN to determine the configurations of the MOS cells of network N. Also, let $\tilde{u}_1, \ldots, \tilde{u}_{R-1}$ be the maximum permissible functions obtained in Step 2 of Algorithm DIMN during the synthesis of network N, and let $\tilde{u}_R = \tilde{f}$.

Step 1 Set $i = 0$. Initially, let the set $T = \emptyset$.

Step 2 $i = i + 1$. If $i > R$, set T constitutes a sufficient test set for Φ_t of N; terminate the algorithm. Otherwise, go to Step 3.

Step 3 For the pair of functions $\tilde{u}_i$ and u_i:

(a) Determine all minimum false vectors of u_i: $C_1, \ldots, C_k$.

(b) Determine the branch-determinative vertices for $\tilde{u}_i$ with respect to u_i. Corresponding to each minimum false vector, C_j, select one corresponding branch-determinative vertex B_j, $j = 1, \ldots, k$. Let Q^{BDV} denote the set $\{B_1, \ldots, B_k\}$.

(c) Select a complete set of blocking vertices, $Q_j^{BV} = \{A_{j,1}, \ldots, A_{j,s_j}\}$, for each B_j, $j = 1, \ldots, k$.

Step 4 Let $R^{BDV}, R_1^{BV}, \ldots, R_k^{BV}$ be sets of n-dimensional specified vectors of $\tilde{u}_i$ corresponding to sets of $(n + i)$-dimensional vectors of u_i, Q^{BDV}, Q_1^{BV}, $\ldots, Q_k^{BV}$, respectively (i.e., each vector $(a_1, \ldots, a_{n+i})$ for u_j corresponds to the vector $(a_1, \ldots, a_n)$ for $\tilde{u}_i$).

(a) For each vector $B \in R^{BDV}$, determine a lower-supportive chain, $(D_1, \ldots, D_\ell)$, of B with respect to $\widehat{NFS}_n^{i-1}(R, \tilde{f})$, and set $T \Leftarrow T \cup \{B, D_1, \ldots, D_\ell\}$.

(b) For each vector $A \in R_1^{BV} \cup \ldots \cup R_k^{BV}$, determine an upper-supportive chain $(D_1,\ldots,D_\ell)$ for A with respect to $\underline{NFS}_n^{i-1}(R,\tilde{f})$, and set $T \Leftarrow T \cup \{A,D_1,\ldots,D_\ell\}$. Return to Step 2.

Example 7.6 Consider, once again, the function f in Fig. 7.1 and the corresponding irredundant network N of Fig. 7.3 synthesized (in Example 7.2) by Algorithm DIMN to realize f. Suppose a test set for the set of all possible faults in N, Φ_t, smaller than that given by Theorem 7.2 is desired. Let Algorithm TSG be applied for network N and function $\tilde{f} = f$. The expressions for u_1, u_2, u_3 selected in Step 3 of Algorithm DIMN (see Example 7.2) during the synthesis of network N are:

$$u_1 = \overline{x}_3$$

$$u_2 = \overline{x_1x_2 \vee x_4u_1}$$

$$u_3 = \overline{x_2x_4u_1 \vee x_4u_2 \vee u_1u_2}.$$

The maximum permissible functions, $\tilde{u}_1$ and $\tilde{u}_2$, developed in Step 2 of Algorithm DIMN are shown in Fig. 7.2(b) and 7.2(d), respectively. $\tilde{u}_3 = f$. Initially, T is the empty set. In Step 3a of TSG, vector (0010) is found to be the only minimum false vector of u_1. (1111) is found, in Step 3b, to be the only BDV for $\tilde{u}_1$, with respect to u_1. Thus, C_1 is (0010) and B_1 is (1111), and $Q^{BDV} = \{(1111)\}$. In Step 3c, a complete set of BV's for B_1 is selected: $Q_1^{BV} = \{(1000)\}$. In Step 4, $R^{BDV} = \{(1111)\}$, $R_1^{BV} = \{(1000)\}$. A LSC, ((1000), (1010), (1011)), for vertex (1111) is chosen in Step 4a, and an USC, ((1111), (1011), (1010)), for vertex (1000) is chosen in Step 4b. Set T following Step 4 is: $T = \{(1111), (1011), (1010), (1000)\}$. Returning to Step 3 for $\tilde{u}_2$ and u_2, $C_1 = (11000)$ and $C_2 = (00011)$ are found to be the minimum false vectors of

u_2. As discussed in Example 7.4, five vertices are BDV's for $\tilde{u}_2$. Selecting two of these, $B_1 = (11110)$ and $B_2 = (10011)$, to correspond to C_1 and C_2, respectively, the set $Q^{BDV} = \{(11110),(10011)\}$ results. As also discussed in Example 7.4, $\{(10110),(01110)\} = Q_1^{BV}$ is a complete set of BV's for BDV (1110), and $\{(10110),(10001)\} = Q_2^{BV}$ is a complete set of BV's for BDV(10011). In Step 4, $R^{BDV} = \{(1111),(1001)\}$, $R_1^{BV} = \{(1011),(0111)\}$, $R_2^{BV} = \{(1011),(1000)\}$. A LSC for (1111) with respect to $\widehat{NFS}_4^1(3,f)$ is ((1011)); a LSC for (1001) is ((1000)); an USC for (1011) with respect to $\underline{NFS}_4^1(3,f)$ is ((1111)); an USC for (0111) is ((1111)); an USC for (1000) is ((1001)). (These selections can be easily verified by the use of Fig. 7.2(c) in which both $\widehat{NFS}_4^1(3,f)$ and $\underline{NFS}_4^1(3,f)$ are shown.) Following Step 4, $T = \{(1111),(1011),(1010),(1001),(1000),(0111)\}$. Continuing the algorithm for $\tilde{u}_3$ and u_3, the final set $T = \{(1111),(1101),(1011),(0111),(1100),(1010),(1001),(1000)\}$, (1101) being added as a BDV for $\tilde{u}_3$ with respect to u_3, and (1100) being added as a BV. Set T is a sufficient test set for network N realizing function f. This set contains 50% fewer vectors than the test set given by Theorem 7.2 for Φ_t and even 38% fewer vectors than the test set given by Theorem 7.1 for Φ_n, the set of all possible NPOCF's.

The following lemma, Lemma 7.1, is given and demonstrated in [Lai 76] (see Lemma 8.1 of [Lai 76]). It will be used in the proof of Lemma 7.2.

Lemma 7.1 Let $\tilde{f}_1$ and $\tilde{f}_2$ be incompletely specified functions of n variables such that $\tilde{f}_1(A) = \tilde{f}_2(A)$ for every $A \in S_c(\tilde{f}_2)$. Let $(u_1,\ldots,u_{R-1}, f_1)$ be a negative function sequence for $\tilde{f}_1$ (i.e., f_1 is a completion of $\tilde{f}_1$). Then $(u_1,\ldots,u_{R-1},\tilde{f}_2)$, with the same $u_1,\ldots,u_{R-1}$, is a feasible partially specified NFS of length R and degree R - 1.

Lemma 7.2 Let $\tilde{f}_1$ and $\tilde{f}_2$ be two partially specified functions of n variables having the relation:

$$\tilde{f}_1(A) = 1 \text{ if } \tilde{f}_2(A) = 1 \text{ and}$$
$$\tilde{f}_1(A) = 0 \text{ if } \tilde{f}_2(A) = 0,$$

for every vector $A \in V_n$. Then, if $NFS^i_n(R,\tilde{f}_1) = (u_1,\ldots,u_i, u^*_{i+1},\ldots,u^*_{R-1}, \tilde{f}_1)$ is a feasible partially specified negative function sequence of degree i and length R:

(1) $NFS^i_n(R,\tilde{f}_2) = (u_1,\ldots,u_i, u^*_{i+1},\ldots,u^*_{R-1}, \tilde{f}_2)$ is also a feasible partially specified negative function sequence of degree i and length R.

(2) Conditional minimum labellings of the n-cube, corresponding to feasible completions, $\underline{NFS}^i_n(R,\tilde{f}_1) = (u_1,\ldots,u_i, \underline{u}^1_{i+1},\ldots,\underline{u}^1_{R-1}, \underline{f}_1)$ and $\underline{NFS}^i_n(R,\tilde{f}_2) = (u_1,\ldots,u_i, \underline{u}^2_{i+1},\ldots,\underline{u}^2_{R-1}, \underline{f}_2)$, of $NFS^i_n(R,\tilde{f}_1)$ and $NFS^i_n(R,\tilde{f}_2)$, respectively, satisfy:

$$L^{\tilde{f}_1,i}_{mn}(A) \geq L^{\tilde{f}_2,i}_{mn}(A)$$

for every $A \in V_n$.

(3) Conditional maximum labellings of the n-cube, corresponding to feasible completions $\widehat{NFS}^i_n(R,\tilde{f}_1) = (u_1,\ldots,u_i, \hat{u}^1_{i+1},\ldots,\hat{u}^1_{R-1}, \hat{\tilde{f}}_1)$ and $\widehat{NFS}^i_n(R,\tilde{f}_2) = (u_1,\ldots,u_i, \hat{u}^2_{i+1},\ldots,\hat{u}^2_{R-1}, \hat{f}_2)$, of $NFS^i_n(R,\tilde{f}_1)$ and $NFS^i_n(R,\tilde{f}_2)$, respectively, satisfy:

$$L^{\tilde{f}_1,i}_{mx}(A) \leq L^{\tilde{f}_2,i}_{mx}(A)$$

for every $A \in V_n$.

(4) Let $\tilde{u}^1_{i+1}$ be the maximum permissible function for $NFS^i_n(R,\tilde{f}_1)$ and let $\tilde{u}^2_{i+1}$ be the maximum permissible function for $NFS^i_n(R,\tilde{f}_2)$. Then:

$$\tilde{u}^1_{i+1}(A) = 1 \text{ if } \tilde{u}^2_{i+1}(A) = 1 \text{ and}$$

$$\tilde{u}^1_{i+1}(A) = 0 \text{ if } \tilde{u}^2_{i+1}(A) = 0$$

for every vector $A \in V_n$.

<u>Proof</u> First consider part 1 of the lemma. Functions $\tilde{f}_1$ and $\tilde{f}_2$ clearly satisfy the conditions of Lemma 7.1. Hence, if $NFS^i_n(R,\tilde{f}_1)$ is a feasible partially specified NFS, $NFS^i_n(R,\tilde{f}_2)$ must also be one.

Next, part 2 of the lemma will be demonstrated. First, consider vertex $\vec{1}$. Clearly, by Step 1 of Algorithm CMNL, $L^{\tilde{f}_1,i}_{mn}(\vec{1}) \geq L^{\tilde{f}_2,i}_{mn}(\vec{1})$. Now suppose that for every vector A of weight j, $L^{\tilde{f}_1,i}_{mn}(A) \geq L^{\tilde{f}_2,i}_{mn}(A)$. Then for each vector B of weight j-1 by the conditions of Step 3 of Algorithm CMNL, $L^{\tilde{f}_1,i}_{mn}(B) \geq L^{\tilde{f}_2,i}_{mn}(B)$ must hold. By induction, the second statement of the lemma is true.

Part 3 of the lemma can similarly be proved by consideration of Algorithm CMXL.

Finally, consider part 4 of the lemma. By Step 3b of Algorithm MPF, if $\tilde{u}^2_{i+1}(A) = 1$, $\underline{u}^2_{i+1}(A) = \hat{u}^2_{i+1}(A) = 1$ (where $\underline{u}^2_{i+1}$ and $\hat{u}^2_{i+1}$ are from $\underline{NFS}^i_n(R,\tilde{f}_2)$ and $\widehat{NFS}^i_n(R,\tilde{f}_2)$, respectively). Then by parts 2 and 3 of this lemma, and the fact that $u_1,\ldots u_i$ are the same for both $\underline{NFS}^i_n(R,\tilde{f}_2)$ and $\underline{NFS}^i_n(R,\tilde{f}_1)$ as well as both $\widehat{NFS}^i_n(R,\tilde{f}_2)$ and $\widehat{NFS}^i_n(R,\tilde{f}_1)$, $\underline{u}^1_{i+1}(A) = \hat{u}^1_{i+1}(A) = 1$. Therefore, by Step 3b of Algorithm MPF, $\tilde{u}^1_{i+1}(A) = 1$.

It can similarly be shown that $\tilde{u}^1_{i+1}(A) = 0$ if $\tilde{u}^2_{i+1}(A) = 0$.

Q.E.D.

<u>Theorem 7.4</u> The set of vectors, T, resulting from Algorithm 7.5 (TSG) for a function $\tilde{f}$ and a network N synthesized by Algorithm DIMN to realize $\tilde{f}$, constitutes a sufficient test set for all possible single and multiple faults in N.

Proof As previously discussed, it is sufficient to show that, for function $\tilde{f}'$ defined as:

$$\tilde{f}'(A) = \tilde{f}(A) \text{ for every } A \in T \text{ and}$$

$$\tilde{f}'(A) = *, \text{ otherwise,}$$

Algorithm DIMN can synthesize the same network N for function $\tilde{f}'$ also. (For convenience, let the synthesis of network N for function $\tilde{f}$ by Algorithm DIMN be referred to as 'Synthesis DIMN,' and let the synthesis of network N' (to be shown identical to N) for function $\tilde{f}'$ by Algorithm DIMN be referred to as 'Synthesis DIMN'.') Then, by Theorem 7.2, the set of all specified vectors for $\tilde{f}'$, a subset of set T, constitutes a sufficient test set for all faults in network N. Thus, T would constitute a sufficient test set for all faults in N.

For each maximum permissible function $\tilde{u}'_{i+1}$ determined in Step 2 of Synthesis DIMN', it will be shown that an expression for u_{i+1}, a negative completion of $\tilde{u}'_{i+1}$ with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_i$, can be chosen identical to that selected for $\tilde{u}_{i+1}$ in Synthesis DIMN and consistent with the conditions of Step 3 of Algorithm DIMN. The proof will be by induction on u_i: first it will be shown that the same expression for u_1, i.e., the same negative completion with respect to $x_1,\ldots,x_n$, can be chosen in both Synthesis DIMN and Synthesis DIMN'; then assuming identical $u_1,\ldots,u_i$ have been chosen in both Syntheses DIMN and DIMN', it can be shown that the same u_{i+1} can then be chosen for both cases.

Consider now the selection of u_1 in Synthesis DIMN'. The maximum permissible function $\tilde{u}'_1$ is obtained in Step 2 of Algorithm DIMN by the use of Algorithm MPF: $\underline{NFS}^0_n(R,\tilde{f}')$ and $\widehat{NFS}^0_n(R,\tilde{f}')$ are obtained and compared to yield $\tilde{u}'_1$. By part 4 of Lemma 7.2,

$$\tilde{u}_1(A) = 1 \text{ if } \tilde{u}_1'(A) = 1 \text{ and}$$
$$\tilde{u}_1(A) = 0 \text{ if } \tilde{u}_1'(A) = 0.$$

Therefore, any negative completion of $\tilde{u}_1$ with respect to $x_1,\ldots,x_n$ is a negative completion of $\tilde{u}_1'$ with respect to $x_1,\ldots,x_n$. In particular, the negative completion u_1 of $\tilde{u}_1$ selected during Synthesis DIMN is also a negative completion of $\tilde{u}_1'$. Since u_1 is a completely specified negative function with respect to $x_1,\ldots,x_n$, a complemented irredundant disjuctive form of it must be the complement of the disjunction of β-terms of the minimum false vectors of u_1:

$$u_1 = \overline{\beta_1 \vee \cdots\ \beta_{k_1}}. \tag{7.1}$$

If it can be shown that this expression satisfies Step 3 of Synthesis DIMN', i.e., if it can be shown that the deletion of any literal or term from the expression would not result in a negative completion of $\tilde{u}_1'$, then clearly u_1 can be selected by Synthesis DIMN', yielding a cell of the same configuration as the first cell in network N.

Suppose a term β_j, $1 \leq j \leq k_1$, can be deleted from expression (7.1) with the resulting expression, $u_1'' = \overline{\beta_1 \vee \cdots \vee B_{j-1} \vee \beta_{j+1} \vee \cdots \vee \beta_{k_1}}$, still a negative completion of $\tilde{u}_1'$. Let C_j be the minimum false vector of u_1 corresponding to β-term β_j. Then the BDV, B_j, corresponding to C_j and all of the vectors corresponding to vertices in its LSC are elements of set T chosen in Algorithm TSG. By the definition of a BDV, B_j must be a specified false vector for $\tilde{u}_1$, i.e., $L_{mn}^{\tilde{f},0}(B_j) = L_{mx}^{\tilde{f},0}(B_j) = 0$. By the properties of a LSC, $L_{mx}^{\tilde{f}',0}(B_j) = 0$ for $\tilde{f}'$, and B_j must also be a specified false vector of $\tilde{u}_1'$. Since the only minimum false vector of u_1 which is less than BDV B_j is C_j, removing the β-term β_j from expression (7.1) would result in B_j being a true vector of u_1''. Thus,

u_1'' can not be a completion of $\tilde{u}_1'$. Hence, no terms can be deleted from expression (7.1) without resulting in a function which is not a negative completion of $\tilde{u}_1'$ with respect to $x_1,\ldots,x_n$.

Next, suppose a literal, x_{j_ℓ}, could be deleted from a β-term, $\beta_j = x_{j_1} \ldots x_{j_r}$, of expression (7.1) with the resulting expression, u_1'', still a completion of $\tilde{u}_1'$. Let C_j be the minimum false vector of u_1 corresponding to β-term B_j. Then, in set T chosen in Algorithm TSG, there must exist a corresponding BDV for u_1, B_j, and vectors corresponding to a complete set of BV's for B_j and all of the vertices in their USC's.

By the definition of a BV, say vector A, it must be a specified true vector for $\tilde{u}_1$, i.e., $L_{mn}^{\tilde{f},0}(A) = L_{mx}^{\tilde{f},0}(A) = 1$. By the properties of an USC, $L_{mn}^{\tilde{f}',0}(A) = 1$ for $\tilde{f}'$, and A must also be a specified true vector of $\tilde{u}_1'$.

Removing one or more literals, x_{j_ℓ}, from β-term B_j in expression (7.1) would mean that a vector E, $E < C_j$, exists which is a false vector for u_1''. However, by definition of complete set of BV's, there exists a BV A for B_j such that $A > E$. Hence there exists a vector A such that $\tilde{u}_1'(A) = 1$, $u_1''(E) = 0$, and $A > E$. This contradicts the assumption that u_1'' is a negative completion of $\tilde{u}_1'$. Therefore, no literals can be deleted from expression (7.1) without realizing a function which is not a negative completion of $\tilde{u}_1'$.

Similarly, it can be shown, for functions $\tilde{u}_2',\ldots,\tilde{u}_R'$, that the same negative completions $u_2,\ldots,u_R$ can be chosen in Synthesis DIMN' as were chosen in Synthesis DIMN. Thus, the identical network N can be obtained by Synthesis DIMN' as obtained by Synthesis DIMN.

Q.E.D.

8. CONCLUSIONS

Some theoretical properties related to MOS cell/negative gate networks were discussed in Section 3. Certain of the properties can be employed in identifying certain non-optimal networks, and some of the properties may be useful in the development of further synthesis algorithms.

Section 4 discussed and compared published MOS network synthesis methods. In particular, a comparison of the approach of [Liu 72] with that of [NTK 72] yielded a new set of synthesis algorithms paralleling those of [Liu 72]. In addition, still other synthesis algorithms based on a new concept of 'clustering' were proposed which employ theidea of [Liu 72] enabling a relatively simple determination of cell configurations in the synthesized network. These new algorithms, however, often produce networks of fewer FET's than those generated by corresponding algorithms of [Liu 72], though they involve somewhat more extensive calculations than those of [Liu 72].

In the preceding sections, several methods for the design of G-minimal MOS networks were discussed. Since a single negative gate is very powerful (in the sense that it can realize any negative function), the implementation of of G-minimal negative gate networks as MOS networks tends to become impractical as the number of variables involved grows beyond a relatively small number. When such impractical designs are produced, the logic designer will find the transformations of Section 5 useful in reducing cell complexities, though no systematic method of applying these transformations was given due to the large number of different possible objectives motivating their use.

A new approach to MOS network design was proposed in Section 6. In this approach, networks are designed by the transformation of existing designs, though this approach does not guarantee optimal networks (except in certain cases in which the pre-existing networks are known to be optimal). The approach seems useful for network synthesis for small values of n (i.e., the number of external variables), or for the simplification of previously designed networks, for large or small n, consisting of cells of a practical size. Certain improvements of, extensions of, and additions to the program implementing the transduction procedures were discussed which could lead to better results and new applications.

Section 7 proposed a new method of generating sufficient test sets for the sets of all possible single and multiple faults in irredundant MOS networks designed by the algorithm of [Lai 76]. This method operates most efficiently when incorporated into the synthesis algorithm of [Lai 76]. An example was given which showed a large percentage difference in the size of a test set for a network compared with the size of the previously best known test set.

REFERENCES

[Ake 68] S. B. Akers, Jr., "On maximum inversion with minimum inverters," IEEE Trans. Comput., vol. C-17, pp. 134-135, Feb. 1968.

[Ake 73] S. B. Akers, Jr., "Universal test sets for logic networks," IEEE Trans. Comput., vol. C-22, pp. 835-839, Sept. 1973.

[BILM 69] C. R. Baugh, T. Ibaraki, T. K. Liu, and S. Muroga, "Optimum network design using NOR and NOR-AND gates by integer programming," Report No. 293, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., Jan. 1969.

[Cul 75] J. N. Culliney, "Program manual: NOR network transduction based on connectable and disconnectable conditions (Reference manual of NOR network transduction programs NETTRA-G1 and NETTRA-G2)," Report No. UIUCDCS-R-75-698, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., Feb. 1975.

[CLK 74] J. N. Culliney, H. C. Lai, and Y. Kambayashi, "Pruning procedures for NOR networks using permissible functions (Principles of NOR network transduction programs NETTRA-PG1, NETTRA-P1, and NETTRA-P2)," Report No. UIUCDCS-R-74-690, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., Nov. 1974.

[Gil 54] E. N. Gilbert, "Lattice theoretic properties of frontal switching functions," J. Math. Phys., vol. 33, 57-67, Apr. 1954.

[Har 65] M. A. Harrison, Introduction to Switching and Automata Theory, McGraw-Hill, 1965.

[Hoh 55] F. E. Hohn, "A matrix method for the design of relay circuits," IRE Trans. Cir. Th., vol. CT-2, pp. 154-161, 1955.

[HS 55] F. E. Hohn and L. R. Schissler, "Boolean matrices and the design of combinational relay switching circuits," Bell Sys. Tech. J., vol. 34, pp. 177-202, 1955.

[Iba 71] T. Ibaraki, "Gate-interconnection minimization of switching networks using negative gates," IEEE Trans. Comput., vol. C-20, pp. 698-706, June 1971.

[IM 69] T. Ibaraki and S. Muroga, "Minimization of switching networks using negative functions," Report No. 309, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., Feb. 1969.

[IM 71] T. Ibaraki and S. Muroga, "Synthesis of networks with a minimum number of negative gates," IEEE Trans. Comput., vol. C-20, pp. 49-58, Jan. 1971.

[KC 76] Y. Kambayashi and J. N. Culliney, "NOR network transduction procedures based on connectable and disconnectable conditions (Principles of NOR network transduction programs NETTRA-G1 and NETTRA-G2)," to be published as a report, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill.

[KM 76] Y. Kambayashi and S. Muroga, "Network transduction based on permissible functions (General principles of NOR network transduction NETTRA programs)," Report No. UIUCDCS-R-76-804, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., June 1976.

[KLM tbp] Y. Kambayashi, H. C. Lai and S. Muroga, "Transformations of NOR networks," to be published as a report, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill.

[KLCM 75] Y. Kambayashi, H. C. Lai, J. N. Culliney, and S. Muroga, "NOR network transduction based on error-compensation (Principles of NOR network transduction programs NETTRA-E1, NETTRA-E2, and NETTRA-E3)," Report No. UIUCDCS-R-75-737, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., June 1975.

[Kau 61] W. H. Kautz, "The realization of symmetric switching functions with linear-input logical elements," IRE Trans. Electron. Comput., vol. EC-10, pp. 371-378, Sept. 1961.

[Lai 75] H. C. Lai, "Program manual: NOR network transduction by generalized gate merging and substitution (Reference manual of NOR network transduction programs NETTRA-G3 and NETTRA-G4)," Report No. UIUCDCS-R-75-714, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., April 1975.

[Lai 76] H. C. Lai, "A study of current logic design problems: Part I, Design of diagnosable MOS networks; Part II, Minimum NOR (NAND) networks for parity functions of an arbitrary number of variables; Part III, Minimum parallel binary adders with NOR (NAND) gates and their extensions to networks consisting of carry-save adders," to be published as a report, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill.

[LC 74] H. C. Lai and J. N. Culliney, "Program manual: NOR network pruning procedures using permissible functions (Reference manual of NOR network transduction programs NETTRA-PG1, NETTRA-P1, and NETTRA-P2)," Report No. UIUCDCS-R-74-686, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., Nov. 1974.

[LC 75] H. C. Lai and J. N. Culliney, "Program manual: NOR network transduction based on error compensation (Reference manual of NOR network transduction programs NETTRA-E1, NETTRA-E2, and NETTRA-E3)," Report No. UIUCDCS-R-75-732, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., June 1975.

LK 75] H. C. Lai and Y. Kambayashi, "NOR network transduction by generalized gate merging and substitution (Principles of NOR network transduction programs NETTRA-G3 and NETTRA-G4)," Report No. UIUCDCS-R-75-728, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., June 1975.

Law 64] E. L. Lawler, "An approach to multilevel Boolean minimization," J. ACM, vol. 11, no. 3, pp. 283-295, July 1964.

Liu 72] T. K. Liu, "Synthesis of logic networks with MOS complex cells," Report UIUCDCS-R-72-517, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., May 1972.

Liu 75] T. K. Liu, "Synthesis algorithms for 2-level MOS networks," IEEE Trans. Comput., vol. C-24, pp. 72-79, Jan. 1975.

LNM 74] H.C. Lai, T. Nakagawa, and S. Muroga, "Redundancy check technique for designing optimal networks by branch-and-bound method," Int. J. of Comp. and Inf. Sci., vol. 3, no. 3, pp. 251-271, Sept. 1974.

Mar 58] A. A. Markov, "On the inversion complexity of a system of functions," J. ACM, vol. 5, pp. 331-334, Oct. 1958; translated from Doklady Akad. Nauk SSSR, vol. 116, pp. 917-919, 1957.

Mor 72] Y. Moriwaki, "Synthesis of minimum contact networks based on Boolean polynomials and its programming on a digital computer," R. of Insti. of Indus. Sci., vol. 21, no. 6, Univ. of Tokyo, Tokyo, Japan, Mar. 1972.

Mul 58] D. E. Muller, "Minimizing the number of NOT elements in combinational circuits," memorandum, Bell Tel. Lab., 1958.

Mur 71] S. Muroga, Threshold Logic and Its Applications, Wiley-Interscience, 1971.

NTK 72] K. Nakamura, N. Tokura, and T. Kasami, "Minimal negative gate networks," IEEE Trans. Comput., vol. C-21, pp. 5-11, Jan. 1972.

Pai 73] M. R. Paige, "Synthesis of diagnosable FET networks," IEEE Trans. Comput., vol. C-22, pp. 513-516, May 1973.

Sha 38] C. E. Shannon, "A symbolic analysis of relay and switching circuits," Trans. Amer. Inst. Elec. Eng., vol. 57, pp. 713-723, 1938.

Sha 49] C. E. Shannon, "The synthesis of two-terminal switching circuits," Bell Sys. Tech. J., vol. 28, pp. 59-98, 1949.

Shi 72] T. Shinozaki, "Computer program for designing optimal networks with MOS gates," Report No. UIUCDCS-R-72-502, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., Apr. 1972.

[SK 69] G. L. Schnable and R. S. Keen, Jr., "Failure mechanisms in large-scale integrated circuits," IEEE Trans. Elec. Dev., vol. ED-16, pp. 322-332, Apr. 1969.

[Spe 69] R. F. Spencer, Jr., "Complex gates in digital system design," IEEE Comp. Grp. News, pp. 47-56, Sept. 1969.

[Yam 76] K. Yamamoto, "Design of irredundant MOS networks: A program manual for the design algorithm DIMN," Report No. UIUCDCS-R-76-784, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., Feb. 1976.

VITA

Jay Niel Culliney was born in Bethlehem, Pennsylvania in 1947. At the University of Illinois, he completed his B.S. degree in Liberal Arts and Sciences and his M.S. degree in Computer Science in 1969 and 1971, respectively. From 1969 to 1976 he was employed as a research assistant in the Department of Computer Science at the university. He is currently employed by Rockwell International in the area of MOSLSI design.

Jay Niel Culliney is a member of Sigma Xi and the Association for Computing Machinery.

BLIOGRAPHIC DATA EET	1. Report No. UIUCDCS-R-77-851	2.	3. Recipient's Accession No.

le and Subtitle

TOPICS IN MOSFET NETWORK DESIGN

5. Report Date
February 1977

6.

uthor(s)
. N. Culliney

8. Performing Organization Rept. No. UIUCDCS-R-77-851

erforming Organization Name and Address
partment of Computer Science
niversity of Illinois at Urbana-Champaign
rbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.
NSF DCR73-03421

ponsoring Organization Name and Address
ational Science Foundation
00 G Street, N.W.
shington, D.C. 20550

13. Type of Report & Period Covered
Technical

14.

pplementary Notes

Abstracts This paper addresses several topics in the logic design of feed-forward (i.e.,
oop-free) networks of MOS cells (gates) to realize specified (sets of) output functions:
ne section presents properties of optimal networks of MOS cells or negative gates
bstract MOS cells) and some properties of interest to logic designers which apply
o non-optimal networks as well. In another section, new optimal network synthesis
ethods based on a redefined concept of 'clusters' are proposed to obtain improved
nthesis results. A third section discusses transformations of MOS networks which
re based only upon the configurations of the networks and the individual MOS cells.
new approach to negative gate network synthesis is given in a fourth section. This
pproach employs more powerful 'transduction procedures' to modify and simplify ex-
sting networks. Actual computer program implementations of these procedures are dis-
ussed. A final section presents a method for the generation of reduced test sets for
e diagnosis of irredundant MOS networks synthesized by the algorithm of Lai.

Key Words and Document Analysis. 17a. Descriptors

ogic design, logic curcuits, logical elements, programs (computers).

b. Identifiers/Open-Ended Terms

Optimal, MOS, FET, design, two-level, multiple-level, cluster, stratified structure, transformation, transduction, irredundant, test set, test set generation, diagnosis, diagnosable

c. COSATI Field/Group

Availability Statement Release Unlimited	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 329
	20. Security Class (This Page UNCLASSIFIED	22. Price